NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DEVELOPING A STANDARD UNIT-LEVEL OBJECT MODEL

by

Arthur L. Cotton, III

September 1997

Thesis Advisor: Arnold H. Buss

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1997	Master's Thesis

4. TITLE AND SUBTITLE
DEVELOPING A STANDARD UNIT LEVEL OBJECT MODEL

5. FUNDING NUMBERS

6. AUTHOR(S)
Cotton, Arthur L.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army TRADOC Analysis Center
Monterey, CA 93943

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release, distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This thesis describes the development of a standard unit-level object model for combat simulations. This thesis is part of an Army Modeling and Simulation Office (AMSO) sponsored study examining selected models from existing and future simulations in order to provide examples and insights to support object standards development. Object models are a key feature of the Department of Defense (DOD) High Level Architecture (HLA) and the Defense Modeling and Simulation Office (DMSO) Conceptual Model of the Mission Space (CMMS). Developing standard objects helps promote consistency among Army combat models and foster both interoperability and model reuse.

As a basis for developing a standard unit-level object model, three legacy and two developmental simulations models were studied. The set of common attributes and methods from the object models of Modular Semi-Automated Forces (ModSAF), Integrated Theater Engagement Model (ITEM), Eagle, WARSIM 2000, and Joint Warfare System (JWARS) were examined for common attributes and behaviors.

The standard unit-level object model and its components were based on the core competencies of military units: planning, communicating, command and control, shooting, movement, and sustainment. This model achieves interoperability by establishing a minimum/essential set of components, attributes, and methods. Finally reuse is maximized through polymorphic component based design.

14. SUBJECT TERMS	15. NUMBER OF PAGES
OMT, Object Models, Combat Models, Simulations	83
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited.

DEVELOPING A STANDARD UNIT-LEVEL OBJECT MODEL

Arthur L. Cotton, III
Lieutenant Commander, United States Navy
B.A., Case Western Reserve University, 1985

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1997

Author: _____
Arthur L. Cotton, III

Approved By: _____
Arnold Buss, Thesis Advisor

_____
Leroy A. Jackson, Second Reader

_____
Richard E. Rosenthal, Chairman
Department of Operations Research

ABSTRACT

This thesis describes the development of a standard unit-level object model for combat simulations. This thesis is part of an Army Modeling and Simulation Office (AMSO) sponsored study examining selected models from existing and future simulations in order to provide examples and insights to support object standards development. Object models are a key feature of the Department of Defense (DOD) High Level Architecture (HLA) and the Defense Modeling and Simulation Office (DMSO) Conceptual Model of the Mission Space (CMMS). Developing standard objects helps promote consistency among Army combat models and foster both interoperability and model reuse.

As a basis for developing a standard unit-level object model, three legacy and two developmental simulations models were studied. The set of common attributes and methods from the object models of Modular Semi-Automated Forces (ModSAF), Integrated Theater Engagement Model (ITEM), Eagle, WARSIM 2000, and Joint Warfare System (JWARS) were examined for common attributes and behaviors.

The standard unit-level object model and its components were based on the core competencies of military units: planning, communicating, command and control, shooting, movement, and sustainment. This model achieves interoperability by establishing a minimum/essential set of components, attributes, and methods. Finally reuse is maximized through polymorphic component-based design.

TABLE OF CONTENTS

LIST OF FIGURES

EXECUTIVE SUMMARY

As the country's principal user of modeling and simulation technology, the Department of Defense (DOD) has a keen interest in promoting reuse and interoperability among simulations to improve efficiency and consistency. Previous efforts to model force and weapon design, material acquisition, and training using procedural programming have been somewhat fragmented and proprietary. These shortcomings coupled with poor documentation required developers of new simulations to start from scratch.

The predominant simulation paradigm, object-oriented programming, models the relationships among objects rather than the procedures used to accomplish objectives. The prospect of reusing object-oriented code for future simulations saves time and money, changing the focus to verifying the code that is reused. Consistent modeling of battlespace entities and phenomena are achieved as libraries of verified object-oriented code becomes available.

The Defense Modeling and Simulation Office (DMSO) was created to coordinate modeling and simulation policy for DOD. The primary objective of DMSO and its Army counterpart, the Army Modeling and Simulation Office (AMSO), is to develop a common technical framework for all simulation models. This research is part of that effort.

This thesis describes the development of an object-oriented standard unit level object model for combat simulations. Standard object models promote interoperability among simulations by providing common names and interfaces through which objects can communicate. Standard object models also help enable code and model reuse as well as the ability to easily incorporate new objects and algorithms into existing simulations.

As a basis for developing a standard unit-level object model, three legacy and two developmental simulations models were studied. The set of common attributes and methods from the object models of Modular Semi-Automated Forces (ModSAF), Integrated Theater Engagement Model (ITEM), Eagle, WARSIM 2000, and Joint Warfare System (JWARS) were examined for common attributes and behaviors.

In order to promote design flexibility, the perspective adopted is that of component- based modeling. The developed model is programming language independent and minimal in design to permit maximum implementation flexibility. The primary standards are specified for components of entities rather than the entities themselves. The ability to assemble the components into entities in different ways simultaneously increases both design flexibility and object reuse. The standard unit-level object model and its components were based on the core competencies of military units: planning, communicating, command and control, shooting, movement, and sustainment.

This research indicates that the development of a standard unit-level object model is most beneficial in bridging standard algorithms and standard data (Functional Description of the Battlespace (FDB)). The standard unit-level object model provides the interface that allows analysis and validation of the standard algorithms and the FDB which is the repository for the data required to support those algorithms.

I. INTRODUCTION

This thesis describes the development of a standard unit-level object model for combat simulations. Many DOD legacy combat simulations are written in various procedural programming languages. Developers of new simulations typically started from scratch since much of the legacy simulation code was fragmented and not designed to interoperate with other models. This changed with the development of object-oriented programming and the prospect of building libraries of interoperable and reuseable standard objects and algorithms. Standard object models promote interoperability among simulations by providing common names and interfaces through which objects can communicate. Standard object models also help enable code and model reuse as well as the ability to easily incorporate new objects and algorithms into existing models. This thesis explores the creation of such a standard object and describes issues, results, and conclusions directed toward its use in future defense applications.

This thesis is part of an Army Modeling and Simulation Office (AMSO) sponsored study examining selected models from existing and future simulations in order to provide examples and insights to support object standards development (AMSO, 1997). Object models are a key feature of the Department of Defense (DOD) High Level Architecture (HLA) and the Defense Modeling and Simulation Office (DMSO) Conceptual Model of the Mission Space (CMMS). A set of standard objects will help maintain consistency among Army models and simulations and foster both

interoperability and model reuse. This thesis contributes to the effort to produce a collection of standard Army objects.

The first section of this chapter introduces the reader to the modeling and simulation technical framework, object oriented modeling concepts, and contrasts unit, or theater simulation models with platform level simulation models. The final section of this chapter provides a detailed statement of thesis.

A. BACKGROUND

As the country's principal user of modeling and simulation technology, the DOD has a keen interest in promoting reuse and interoperability among simulations. A robust standard unit-level object model is required to obtain the benefits of standardization. Robustness of the standard unit-level object model will be achieved by defining the minimum essential attributes and methods of the standard unit object. Further, this will permit the model to be partitioned into components with well-defined interfaces. By taking advantage of common interfaces, the standard unit-level object will allow the analyst to evaluate algorithms (attrition algorithms, for example) and select those which offer the most training or analysis benefits. This same feature will permit a standard unit-level object to be used in either theater level or high resolution models without special considerations.

1. Modeling and Simulation Technical Framework

DOD Management Directive 5000.59 and the resulting DOD Modeling and Simulation (M&S) Master Plan developed several objectives to make the M&S community successful. The immediate objective is to build a common technical

framework for M&S. Included in this effort are a High Level Architecture and a Conceptual Model of the Mission Space.

Technological advancements in Distributed Interactive Simulation (DIS) led to the DMSO common technical framework. A precursor to HLA, DIS permits geographically disperse simulations to interoperate via either a Local Area Network (LAN) or a Wide Area Network (WAN). DIS links simulations from different services to improve joint warfare training and analysis in a realistic "virtual battlefield environment".

a) High Level Architecture

The Defense Modeling and Simulations Office (DMSO) directed the development of the High Level Architecture (HLA) in order to facilitate interoperability among simulations and promote reuse of simulations and their components. All DOD simulations are to comply with the powerful and flexible HLA communication infrastructure by October 1, 2001. (USD, 1996) HLA achieves interoperability among simulations by specifying a Run Time Infrastructure (RTI) to exchange data while attempting to avoid bandwidth and CPU limitations. HLA promotes reuse by specifying representation of individual simulations and groups of simulations through the use of the Object Model Template (OMT). When groups of individual simulations are connected through a network using a RTI they are said to be a federation. The OMT is the object model description of the federation, Federation Object Model (FOM), and simulations, Simulation Object Models (SOM). The HLA OMT requires an object class structure table, an object interaction table, an attribute/parameter table, and a data dictionary. The standard unit-level object model must also meet these HLA OMT requirements in order to be reuseable and interoperable. The SOM includes a component structure table, an

object associations table, and an object model metadata file (a log of the developmental history of the SOM and specific execution characteristics of the simulation).

b) CMMS/FDB

The CMMS is a simulation-independent first order abstraction of the real world for activities associated with a particular set of missions. There will be several Conceptual Models of the Mission Space corresponding to broad mission areas such as conventional combat operations, other military operations, acquisition, and analysis. The mission space structure, tools and resources will permit development of consistent, interoperable, and authoritative representations of the environment, systems, and human behavior. The Army's contribution to the CMMS is called the Functional Description of the Battlespace (FDB). The purpose of the FDB is to document the standard descriptions of components and characteristics of battlefield functions. (Blakely, 1996)

2. Object-Oriented Design and Programming

Object-oriented modeling is a method of examining problems based on real-world concepts and phenomena. Even though the object-oriented methodology has been in use for more than a decade, application of object-oriented design in DOD combat simulations is still in its infancy. This is not simply a programming technique, but it is an approach to software design. Combining both data structures (attributes) and behaviors (methods) into a single element, the object is the cornerstone of object-oriented modeling. In contrast to procedural programming in which data structure and behavior are loosely associated, object-oriented designed models better address front-end conceptual issues, rather than back-end implementation issues. An object-oriented development approach encourages software developers to work in terms of the real world domain throughout the

development cycle. Object-oriented designs are very useful in conceptual communication between customers, application experts, and modeling enterprises. Object-oriented design not only allows information to be shared within an application, but also offers the prospect of reusing designs and code on future projects. This power is largely the result of four main aspects which characterize object-oriented design: identity, inheritance, encapsulation, and polymorphism.

Identity is the organization of data and methods into entities called objects. Two objects may have identical attribute values but remain distinct since each is a different instance of a class. The class is the blueprint for the objects, an abstraction providing specifications for the objects and the means to create new objects. Both objects and classes can represent physical entities, such as tank (class) alpha one (object) in the first line of defense, or conceptual entities, such as tank battalion (class) alpha (object) in the command structure. A class enumerates a list of instance variables by type and name which specifies an object's data. Similarly, the class specifies which methods are associated with an object by giving the method a name, arguments, and the return type. An object-oriented language provides a library of standard classes from which the programmer can use as given or modify via inheritance.

Inheritance implies a hierarchical relationship between classes. A superclass contains all of the attributes and methods common to all subclasses which may include additional data or behaviors. Figure 1 depicts the inheritance of Land, Air, Maritime, and Special Operations Forces (SOF) units as abstract units. It further depicts a Ground Maneuver Unit as a Land unit and the multiple inheritance of a Multifunctional unit from land, air, maritime, and SOF units. One common use of inheritance is behavior

refinement. This permits differences in the resolution or specialization between similar classes in which the super-class has desirable behaviors that need to be implemented differently to be properly represented in the subclass. When the subclass alters the inherited super-class method of the same name (and signature), we say that the method is overridden. Another technique to enable different implementations of methods is to

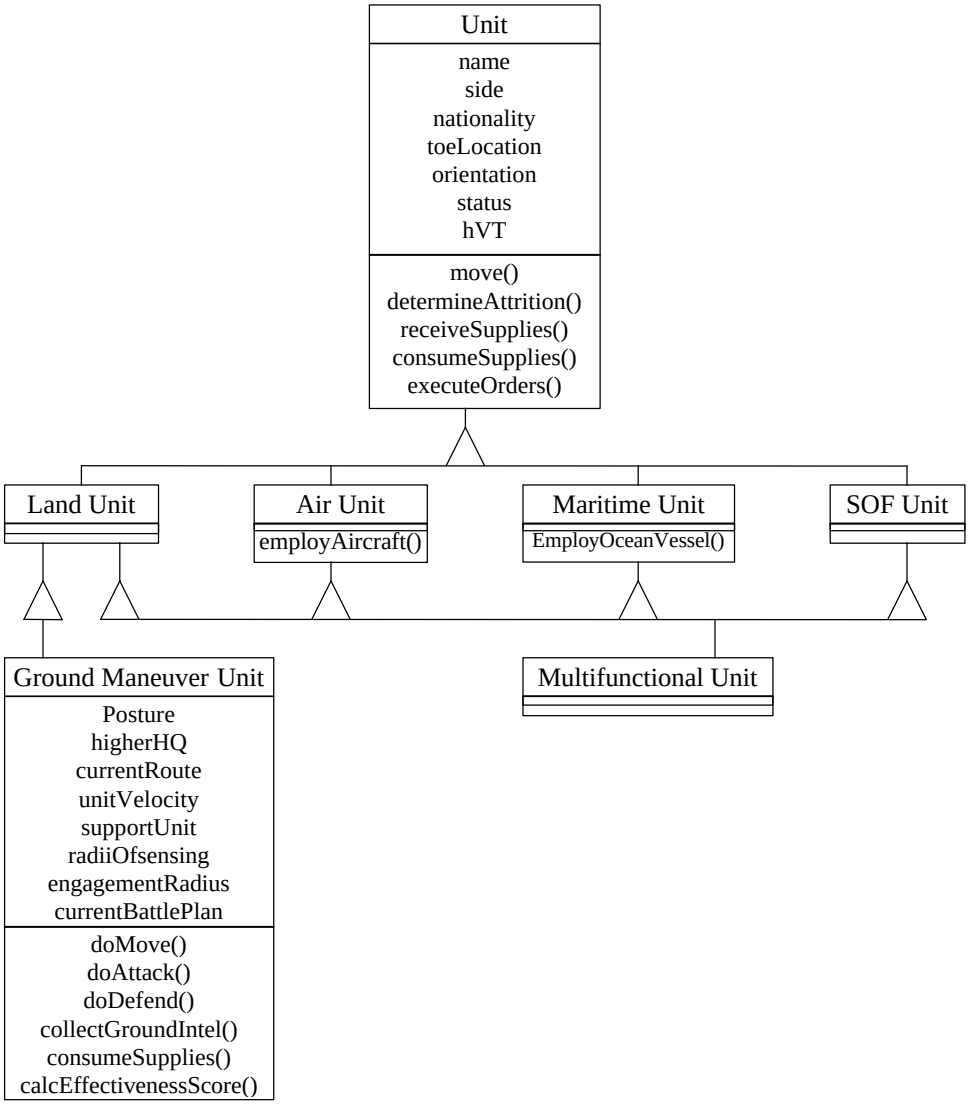


Figure 1: Class Hierarchy

develop abstract methods in the super-class. This ensures that all members of the class hierarchy will respond to the method while permitting each subclass to appropriately specify the implementation of the method.

Inheritance is most appropriately used when the subclass is a “kind-of” the super-class and we wish to succinctly capture the similarities and differences between the classes. When an object has similarities with more than one class, the object may be created using multiple inheritance or alternatively as a single inheritance with multiple interfaces. This advanced object-oriented concept is discussed in detail in Chapter IV Section D. When it is more intuitive to say that an object “contains” or is a “part-of” some set, then aggregation (whole-part relationship) is the more appropriate association. Figure 2 demonstrates the aggregation of two firing platoons (a type of ground maneuver unit) into each of three Field Artillery Batteries (a type of ground maneuver unit), all of whom are a part of a Field Artillery Battalion (also a type of ground maneuver unit). Both inheritance and aggregation are important associations in clearly defining the structural dependency

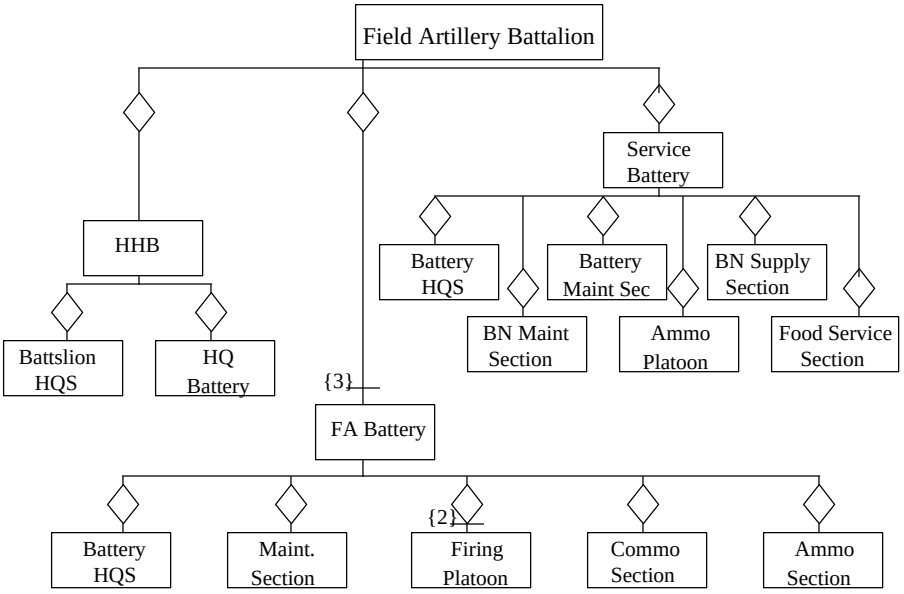


Figure 2: Aggregating Components

between classes. The properties of the super-class need not be repeated in each subclass. This feature greatly reduces duplication within designs and programs.

Encapsulation (or information hiding) is a programming technique which separates an object's public interface from its private data and possibly some of its methods. This technique not only prevents corruption of object attributes but also enhances code reusability and component-level modeling. Encapsulation makes the code more robust by reducing the assumptions required about the other objects in a model, since the internal representation of data is of no interest to the interacting objects. Changes in the internal representations of the objects may be made with minimal impact on the entire model, whereas changes to the public interfaces have more far reaching consequences.

Polymorphism refers to the ability to use the same method name for different methods even within the same class of objects. This permits the subclass object to use the super-class methods in the most appropriate manner for the subclass. The class of the object defines the implementation of the method, relieving the programmer from the requirement to change existing code so long as polymorphic methods are provided for new classes. To illustrate this feature, first consider a ground vehicle object and an airframe object which both invoke a method called moveTo(). The different classes can properly implement the class unique requirements of this method without the user having to separately identify a specific mode of movement for each class.

By designing object-oriented programming to use standard abstract classes and methods, the full benefits of modularity and reuse can be realized. Abstract classes cannot support instantiation of objects without first being subclassed. In this manner, abstract methods serve as place holders for desired traits envisioned for potential subclasses whose formal implementation is not yet known.

3. Standard Objects

There are three major benefits derived from establishing standard objects for use in future combat simulations:

- Enforcing model consistency. By establishing a common set of object names and interfaces, all models, especially those that may be distributed, can consistently treat units and platforms.
- Supporting model development. Standard objects will promote reuse and improve interoperability. Reducing redundant design will improve the

efficiency of the combat modeling community and the probability of producing an unacceptable model is significantly reduced.

- Improving verification and validation. Standard objects will reduce the subjectivity in the verification and validation (V & V) process. The V & V team will be able to evaluate the model on the basis of its compatibility with the standard objects, focusing their primary efforts on the quality of the model.

Developing standard objects has potential pitfalls, however. Establishing an inadequate standard object would lead to an increase in design effort as modelers struggle to overcome unnecessary limitations. Perhaps the greatest danger is in believing that standard objects are timeless. A designer may develop a revolutionary modeling improvement that is inconsistent with a standard object.

4. Unit vs. Platform Level Simulation Models

Combat simulation models have differing levels of resolution depending on the purpose of the model. Simulations designed to improve command and control, analyze force composition, and promote analysis of strategic options are generally written for mid-grade to senior leaders and center around the corps or division level. In these types of simulations, lesser units, their sensors and weapons systems, are aggregated into battalion or company levels for movement and attrition algorithms. Higher resolution simulations model entities down to the individual soldier, vehicle, or aircraft level. While some aggregation is permitted in these models, the aggregation is usually limited to the platoon level. High resolution simulations are most often used for training individual units and for analyzing the effectiveness of new military systems and tactical doctrines. In order to maintain a realistic battlespace domain in high resolution models, individual platforms should be modeled as separate entities. The separation of unit and platform battlespace entities segregates this thesis from other work in this study.

In order to develop a standard unit level object, it is important to define what entity a unit represents. The Unit class is used to represent battlespace entities which direct its components to carry out actions in support of a mission. A unit, then, has a strategic or tactical purpose on the battlefield. This definition is dependent on the resolution of the model. Recall that, depending on the purpose of the simulation, the resolution on the simulation may either be at the platform level, aggregated at the unit level, or mixed. For example, one modeler may elect to model a Patriot Missile Battery with separate platforms for the remote sensors and the missile launch units. This type of

representation may be useful to explicitly model the sensor capability, human recognition, target assignment, and actual prosecution of individual targets. This same Patriot Battery may also be modeled as a composite unit by a modeler whose focus is on aggregated combat models. The flexibility in this design allows the modeler to focus more on the purpose of the simulation than any implementation constraints imposed by the standards.

B. STATEMENT OF THESIS

This thesis develops a standard unit-level object model, demonstrates the flexibility of the design, and describes the rational and methodology for creating a standard unit-level object.

The remainder of this thesis includes a discussion of the methodology of the research in Chapter II, covering the general concept of the research and a method of explaining both legacy and proposed object models. The third chapter contains an analysis of legacy and future simulation object models, highlighting the strengths and weaknesses of each model. The proposed standard unit-level object model and an alternate are presented in Chapter IV. The thesis concludes with supporting arguments for the proposed standard unit-level object model and recommendations for further study.

II. METHODOLOGY

This chapter discusses the methodology used in developing a standard unit-level object model. Also covered are the key aspects of the symbology used in presenting the standard unit-level object model.

As a basis for developing a standard unit-level object model, three legacy and two developmental simulations models were studied. The set of common attributes and methods from the unit-level object models of Modular Semi-Automated Forces (ModSAF), Integrated Theater Engagement Model (ITEM), Eagle, WARSIM 2000, and Joint Warfare System (JWARS) were examined for common attributes and behaviors.

The standard unit-level object model will provide the framework for the conceptual mapping of entity attributes and behaviors in legacy simulations to an object representation. By using a component based design, the standard unit-level object model will provide the flexibility to incorporate the standard unit-level object model into any number of future simulations. The goal for the standard unit-level object model is to capture the realism of the application domain while preserving the flexibility of the modeler to vary the resolution of his model. Further, ensuring that the proposed model is both interoperable and reuseable with simulations from all agencies adds robustness to the model.

A. CONCEPT

A model is an abstraction of a complex system designed to provide a greater level of understanding of the system, its components, and its associations with other systems.

This increased understanding is achieved by omitting nonessential details of the system and by making assumptions about the interactions of the entities to be modeled. Models may take the form of physical or mathematical models. A structural engineer might build a scale mock-up of a bridge to test in a wind tunnel or transportation management may mathematically model routes between shipping hubs and the customers they serve. Blueprints, pencil sketches for paintings, and even outlines for books can be considered models. Besides being cheaper than building and testing complete systems, models and their associated simulations are often safer, provide an analytical evaluation tool, and enable early flaw detection in the proposed design.

Modeling with object-oriented design permits engineers, developers, and customers to communicate clearly the complex abstract concepts and specifications of a system. At the core of object-oriented design is the object. An object is a discrete entity which is distinguishable by a quantized data structure and particular behaviors. Objects may be either concrete, such as a tank or rifle company, or abstract, as in the case of a Ground Combat Unit. In order to take advantage of common structures, objects are said to be grouped into classes. The class is the blueprint for the objects, an abstraction providing specifications for the objects and the means to create new objects.

B. BUILDING THE STANDARD UNIT-LEVEL OBJECT MODEL

The component based approach to developing a standard unit-level object model is best portrayed by a standard graphical representation. Additionally, definitions of the proposed functional organization and associations of the components add clarity to the standard model.

1. Graphical Representation of the Standard Unit-Level Object Model

In order to better communicate the complex features and functions of simulations, the standard unit-level object model must be visually complete and meaningful without being redundant. In this thesis the classes will be depicted using the Unified Modeling Language (UML) notation, emphasizing class hierarchy (inheritance relationships), object attributes and methods, and associations. Like James Rumbaugh's Object Modeling Technique (OMT), UML provides support for modeling classes, objects, and the many kinds of relationships among them, including inheritance, association, and aggregation. UML is itself extensible, allowing modelers to represent either simple or complex systems clearly and succinctly. Basic UML notation is displayed in the Appendix. This methodology employs three types of models to describe a system. The object model depicts the static structure of objects in a system and the relationships that bind the objects. This is the focal point of the thesis. The dynamic model specifies the control and implementation of a system by using state diagrams to show the aspects of the system which change over time. The functional model contains data flow diagrams which describe the data value transformations within a system. (Rumbaugh, 1991)

2. Class Hierarchy

A class hierarchy is comparable to an organization chart, where the subordinate units in the organization inherit certain identities and routines from their parent unit. In the UML, individual classes are represented as outlined rectangles with either one or three boxed compartments. The mandatory compartment label is for the name of the class. When the second and third sections are used, they list the attributes and methods defined by the class. In order to efficiently use space and to avoid over-exposure of the model, a class may be diagrammed only to the class level. This provides some degree of

abstraction above the vast underlying details found in military simulations. For illustrative purposes, consider a possible class hierarchy of a truck as provided in Figure 3. Examples of the abstract truck class include the instantiable private truck class and the abstract commercial truck class.

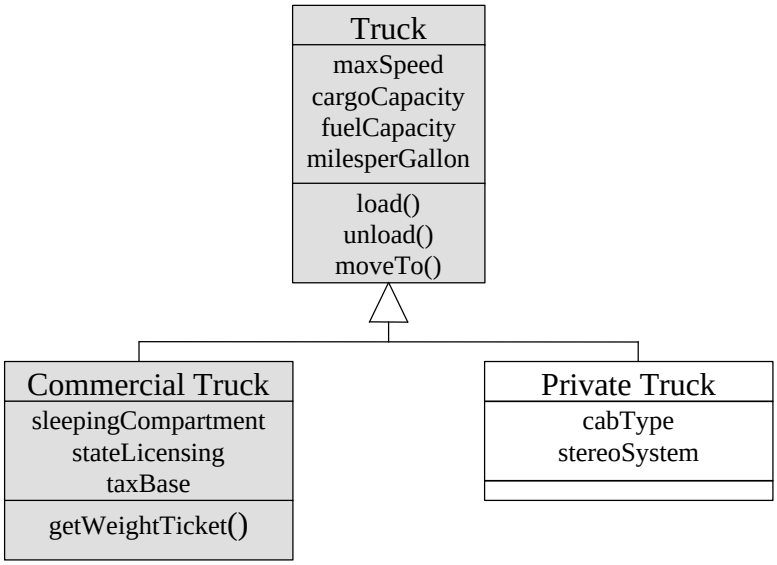


Figure 3: Truck Class Hierarchy

The approach taken in constructing the class hierarchies is to be as abstract and as minimal as possible. Methods and attributes will be represented as high up in the hierarchy as possible. Only public attributes will be explicitly shown as these are considered to be the information which is required to be visible in the model. Subclasses may then override inherited methods for specialization.

3. Attributes

An attribute is a data value held by the objects in a class. From Figure 3 above, the Truck Unit has attributes of maxSpeed, cargoCapacity, fuelCapacity and milesperGallon. In addition to these inherited attributes, the Private Truck class also has a cabType and stereoSystem. Each attribute has a specified value for a particular object instance, but different instantiated objects may or may not have the same value for a given attribute.

When specified, attributes are listed in the first box beneath the class name. Depending on the depth of the presentation of the objects, each attribute may be followed by details, such as type and default value.

To identify attributes consider possessive phrases like “the maximum speed of the truck” or “ the cargo capacity of the truck” in which the attributes correspond to the first noun. Specific values of the attributes would then be adjectives of these nouns, such as one hundred three miles per hour or 644 cubic feet of cargo space. The modeler must capitalize on his knowledge of the application domain to identify attributes. If an independent existence of an entity is more important than its singular value, then the entity should be modeled as an object. Keep in mind that the truck class is presented for illustrative purposes and is not a full application. Actual applications tend to have many more attributes per class than shown in Figure 3.

4. Methods

Transformations or functions that may alter the state of an object are called methods. Methods are common to all objects in its class. Methods, when specified, are listed in the second box following the name of the object. From Figure 3, all trucks can

load, unload, and move to a location while commercial trucks also can get weight tickets. The behavior of the object depends on its class.

Methods which apply to several different classes with differing resultant behaviors are known as polymorphic methods. The modeler must be careful to ensure that polymorphic methods have the same signature in each of its classes. The advantage of this approach is that less information is needed about an object before invoking one of its methods. In particular, the polymorphism of inheritance can be exploited to avoid testing an object for its type.

Methods will be indicated by parentheses () to distinguish them from attributes. Each method may have one or more arguments; the signature of a method is its argument types and its return type. One distinct group of methods are queries. This type of method merely computes a functional value or completes some logical test.

5. Associations

Stand-alone objects are uncommon due to the complex nature of combat modeling. Most models consist of many distinct objects which interact with each other. Association diagrams depict these relationships between objects. Associations are the framework of the standard unit-level object model, providing access paths between objects. Figure 4 expands upon the truck example of Figure 3 to show the association between a truck and its dealer.

Associations may be either bi-directional or uni-directional. Multiplicity indicates how many instances of one class may relate to a single instance of an associated class. In Figure 4, notice that a truck may be sold by one dealer (as indicated by the open circle)

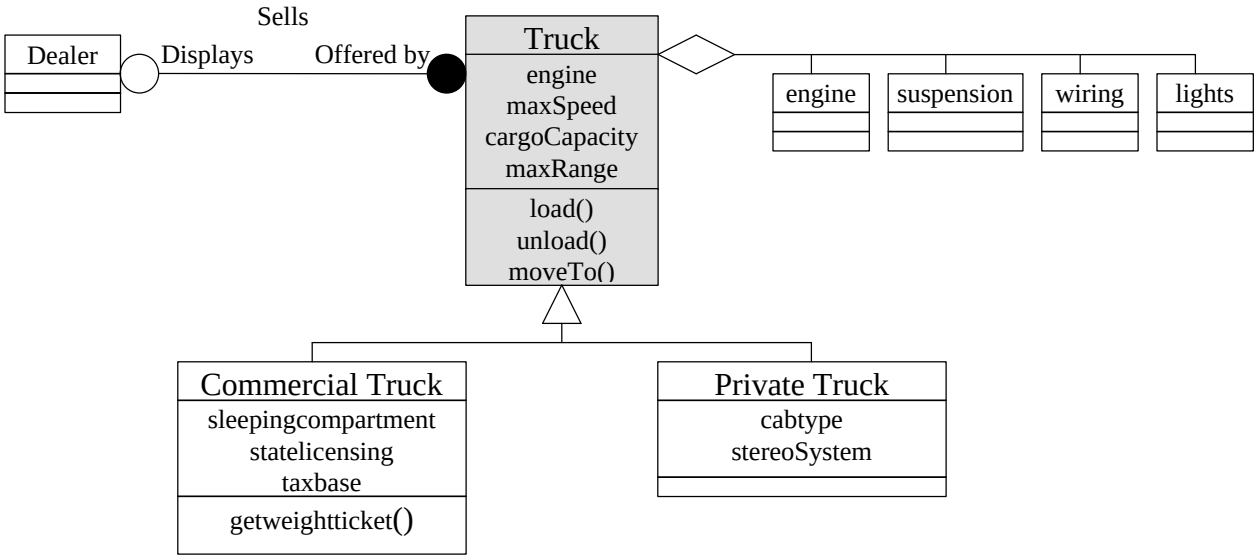


Figure 4: Preliminary Truck Class Diagram

while that same dealer may display numerous trucks for sale (indicated by the filled circle). Associations often correspond to verb phrases including directed actions, ownership, or satisfaction of some type of condition.

To identify associations look for any dependencies between two or more classes. Associations show dependencies between classes at the same level of abstraction as the classes themselves. To preserve design freedom, decisions about implementation of associations should be deferred as long as possible.

6. Aggregations

When objects are comprised of several component objects, the association between the object and its components is called an aggregation. In Figure 4 a truck has an engine, a suspension system, some wiring, and lights. A parts listing from a technical drawing is a compelling example of aggregation. Depending on the application domain, aggregation may be either fixed, variable, or recursive. The most restrictive structure is fixed aggregation where, for example, a truck has exactly one engine and four wheels. A variable aggregate has a finite number of possibilities, but the number of components may

vary. Variable aggregation would relax the requirement for a truck to have exactly four wheels, perhaps allowing an even numbers of wheels between four and eighteen. A recursive aggregate contains an instance of the same kind of aggregate component. Recursive aggregation is exemplified by a military force structure, where brigades are composed of a number of battalions which in turn are composed of a number of companies. The number of potential levels is unlimited.

A goal of the analysis of the object class is to fully specify the application domain without introducing a bias to any particular application. A good design will capture the essential features of the problem without introducing implementation artifacts that prematurely restrict design decisions. The object model provides this detail by showing the static structure of the real world.

Having established common a symbology and phraseology associated with object models, legacy and developmental simulation models can be examined to study structures which may become components of the standard unit-level object model.

III. LEGACY AND DEVELOPMENTAL OBJECT MODELS

This chapter is an analysis of three legacy models, Modular Semi-Automated Forces (ModSAF), Integrated Theater Engagement Model (ITEM), and Eagle and two models currently under development, WARSIM 2000 and the Joint Warfare System (JWARS). Both similarities and differences are discussed with the intent of providing historical and prevailing perceptions of modeled units. The level of information available about these models dictates the depth of the discussion.

A. MODSAF

ModSAF, or Modular Semi-Automated Forces, is the open architecture successor to the SIMNET and ODIN Semi-Automated Forces systems. ModSAF provides uniform methodology and software support for creating and controlling entities within a simulated battlefield. The goal of ModSAF is to replicate the outward behavior of simulated units and their component vehicles and weapon systems to a level of realism sufficient for training and combat development. The breadth of the model is limited to ground and air entities (maritime units are not represented) and the depth ranges from company level to individual vehicle and weapon systems. ModSAF was developed by Loral Advanced Distributed Simulation for the U.S. Army Simulation, Training, and Instrumentation Command (STRICOM) and the Advanced Research Projects Agency - Advanced Systems Technology Office (ARPA-ASTO). ModSAF employs object based design ensuring that the model is Ada compatible, but is not documented using established object-oriented methodology. ModSAF is programmed in C to maximize compatibility

with a variety of hardware platforms and so that run-time greater or equal than real time is achieved. (LORAL, 1995)

ModSAF simulates entities by enabling them to execute a realistic range of basic actions inherent to the entity type. When a unit is simulated, ModSAF not only creates the appropriate entities (plane, tanks, dismounted infantry, etc.) in a unit but also builds a structure corresponding to the unit hierarchy. Figure 5 shows a possible ModSAF unit hierarchy. Instead of single inheritance, ModSAF uses aggregation so larger classes are composed of varying quantities of smaller classes. Commands can then be issued to either the top-level units or to their subordinate units or vehicles. ModSAF's units can take advantage of situational awareness and opportunities for cover and concealment when

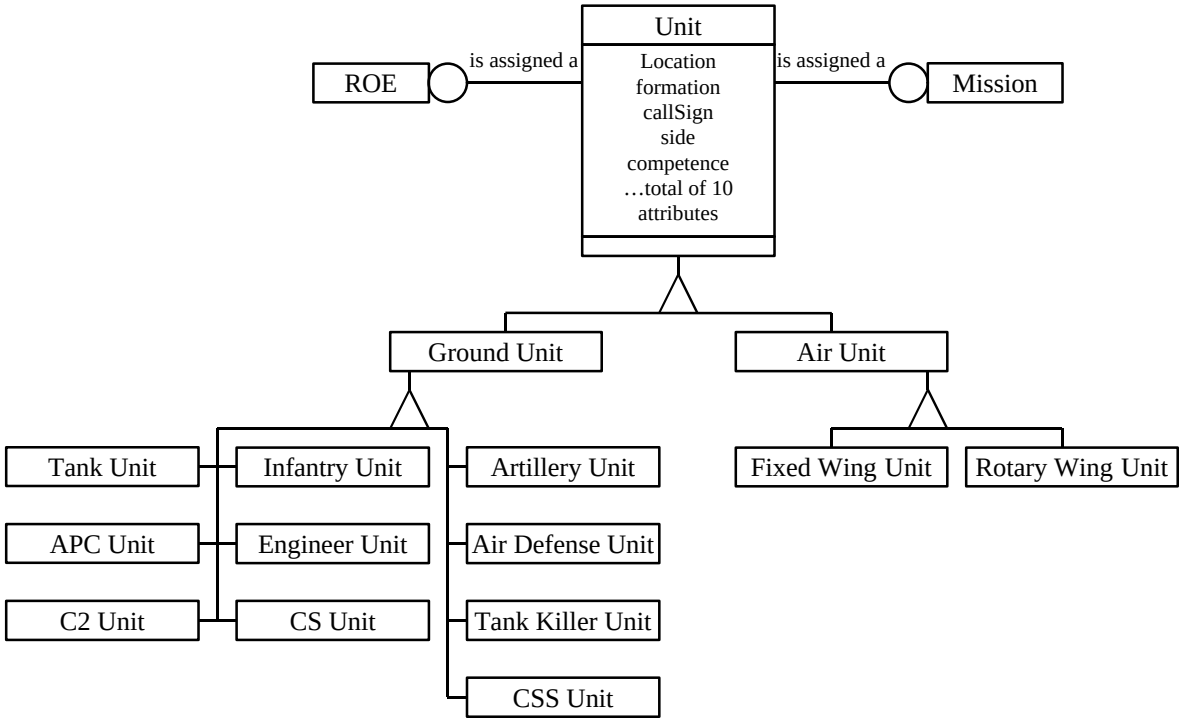


Figure 5: ModSAF Class Hierarchy

they perform tactical movement and combat. Its platoons can perform advanced platoon behaviors and can be lead by a platoon leader in a manned simulator. ModSAF combat service support capabilities give vehicles the ability to repair and resupply other vehicles. The user can also interrupt the current mission to perform new tasks and then return to the original mission.

ModSAF units are aggregates of platforms or subordinate units as appropriate to the level of the simulated unit. In fact, graphical representation of the unit can be displayed at various levels of aggregation ranging from company to platform level (determined by the user). Because the model is primarily a training and combat development model, ModSAF units move, cause attrition, and are attrited at the platform level. The units move in a formation and may even have sub-formations, but each platform is represented independent of its unit. The same reasoning holds during simulated battle between ModSAF units. Individual platforms detect and engage other individual platforms in accordance with its units mission and rules of engagement. Figure 6 depicts a possible object model representation of an instantiated ModSAF M2 reinforced

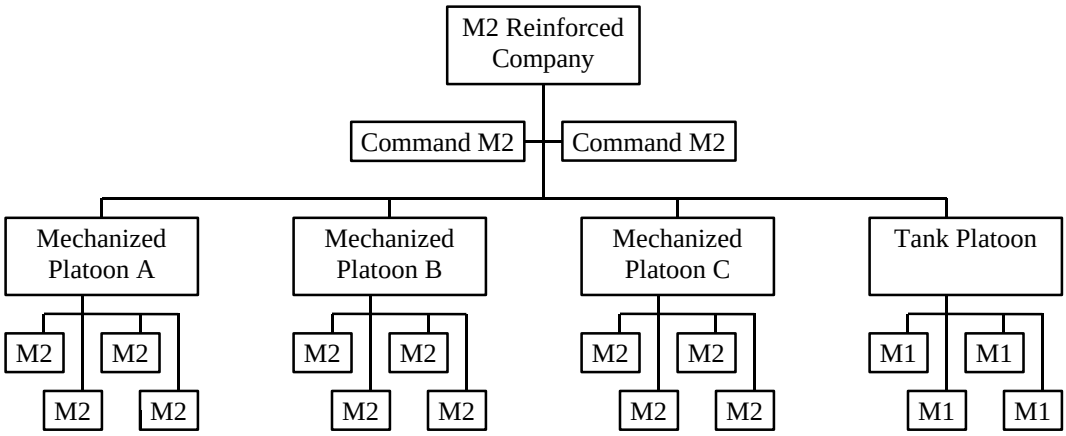


Figure 6: ModSAF M2 Reinforced Company

company. Units are controlled by doctrinally correct tactics involving tasks and missions necessary to perform functions such as move, shoot and communicate, formation keeping, target detection, identification and selection, and fire planning and distribution. ModSAF behavior is controlled by taskframes-a collection of related tasks that run simultaneously. A mission is a network of taskframes connected by enabling tasks, which determines when a condition has been met so that a unit can transition between mission phases. Tasks can be interrupted and altered by the operator, as in the issue of a fragmentary order.

B. ITEM

ITEM is an interactive, two-sided, object-oriented simulation providing integrated air, land, and naval forces for the analysis of joint force operations in theater level campaigns. ITEM is funded by the Defense Nuclear Agency (DNA) and the Department of the Navy (DON). It is principally used by the Commander-in-Chief Pacific (CINCPAC) to model both conventional and nuclear phases of conflict. (Science Applications International Corporation, 1995)

ITEM employs interactive, human decision-making processes for strategic decisions, consequently maintaining a single campaign state. The individual event modules use embedded rules for the tactical decision making and are both multiple-state and automated. Most of the event modules use a Monte Carlo simulation since they model situations which are too complex to be described with deterministic models. In comparison to the TACWAR air/ground campaign model which is single-state and deterministic, ITEM's design is quite innovative.

Ground force combat events in ITEM are conducted every hour of run time. The model moves the force along their paths and computes the results of combat interaction for the current hour using a time step that can be as small as one minute. Movement can be either by force or by unit. If movement by force is specified, the units are moved lock-step parallel to the force path points at the speed of the slowest unit. Combat attrition in ITEM is modeled at the unit level.

Focusing on the ground combat objects, ITEM uses class association to depict a force hierarchy (bottom to top) of components, units, forces, corps, and armies. Figure 7 displays this class association. Examples of components include; tanks, armored fighting vehicles, mobile SAM launchers, combat troops, artillery pieces and transport vehicles.

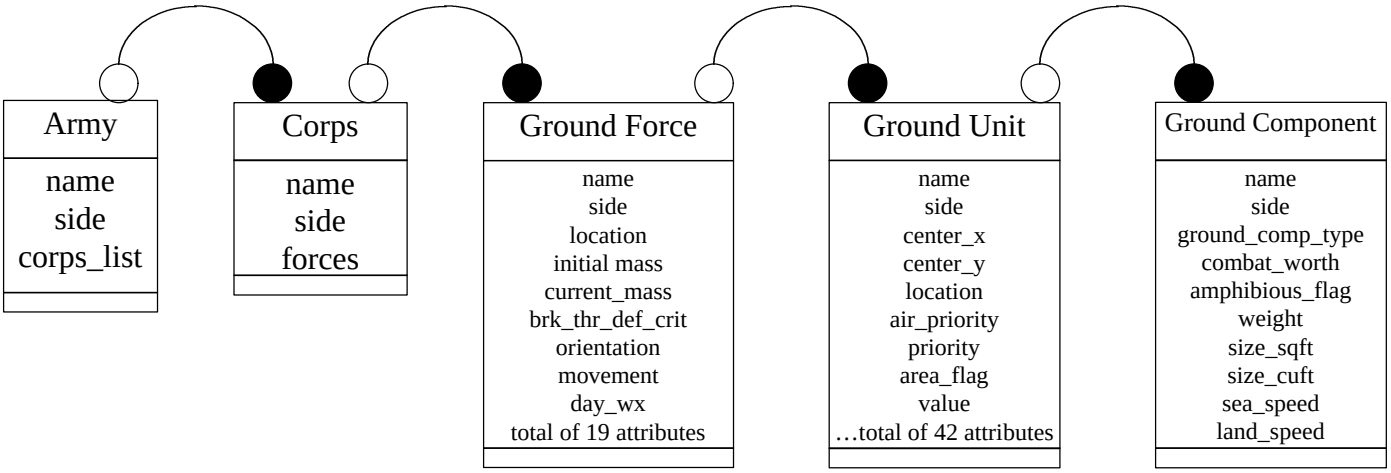


Figure 7: ITEM Ground Combat Class Associations

Components are assigned a relative combat worth in units of tank equivalent mass. Units then are a collection of ground force components whose combat worth is also an aggregate of the combat worth of its components. Units are typically defined as brigades or battalions. Likewise, ground combat forces (model for divisions) are formed from units, corps are formed from forces, and armies are formed from corps. Armies and corps

are used for report generation only, hence the focus of ITEM is at the force level and lower.

The ground unit components in ITEM are used to represent categories of combat elements that can be used to build units. Typical elements defined by the modeler include: tanks, armored vehicles, artillery, trucks, and personnel. Sample Component types are illustrated in Figure 8.

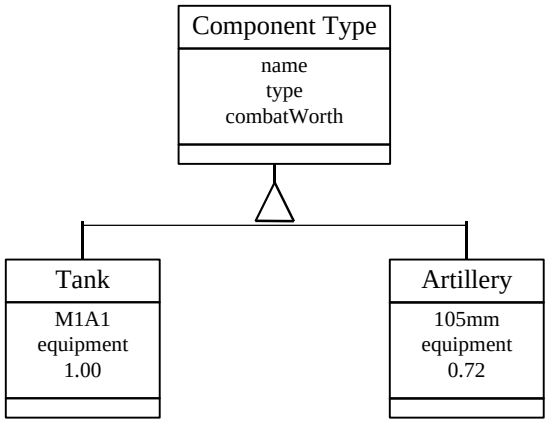


Figure 8: ITEM Component Type Class Structure

The attributes of a component are fixed and do not change as a result of combat engagements. A component type, once created by the user, provides a template for the creation of instances of the component that are created at the time components are assigned to a unit.

The next higher entity in the hierarchy of the ground force objects in ITEM is the unit. The unit is used to represent the smallest tactical unit to be modeled. A partial listing of a unit's attributes are shown in Figure 7 above. The name of the unit uniquely identifies the unit. The location specifies the relative position of the unit within the force. The list of components defines the composition of the unit in terms of equipment and personnel. The combat worth is aggregated as the quantity of a particular component type

times its combat worth. This attribute is useful for analyzing the contribution of each component type on the effectiveness of the unit. The attribute, path points, are objects used to reflect the location and tactical posture of a unit at present and for the future. Both units and ground forces have the pathPoint attribute, however, the associated posture object of the path point object in a force is not operational. Figure 9 depicts the association of the unit attribute pathPoint. Postures have user assigned names and

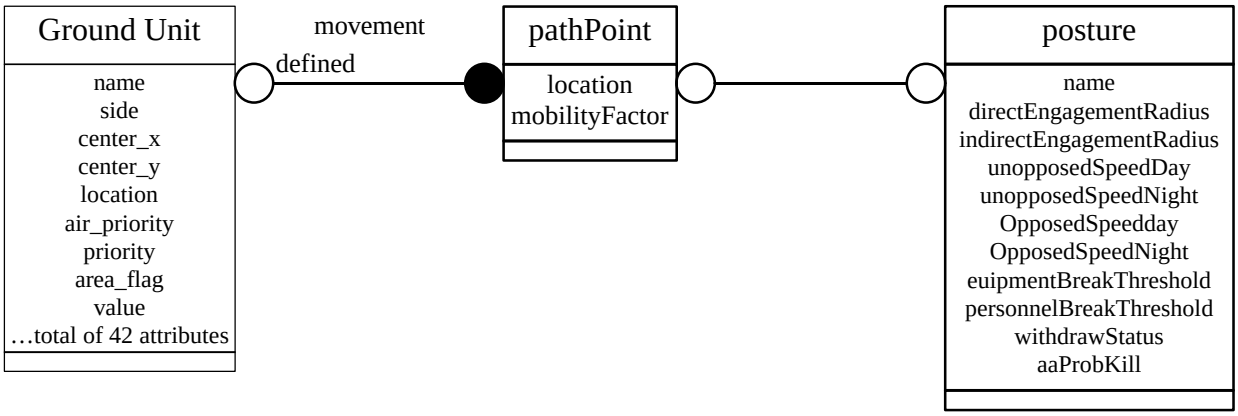


Figure 9: ITEM Unit Association of Path Points

allow the modeler to vary the capabilities of a unit over time as a function of the units assumed tactical disposition and mission. Figure 9 shows that a unit may have several path points which have unique postures associated with them.

The posture object has several attributes designed to be used in determining movement and sensing. The engagement radii define the area centered on the location of the unit in which the unit is assumed to engage opposing units with either direct fire or indirect fire. The various speeds are used by the model to move the unit along the path defined by the path points as a function of the time of day and the tactical posture of the units. The break thresholds are values between 0 and 1 specified by the user to indicate the fraction of the unit's original equipment or personnel mass (combat worth) at which

the unit ceases to engage opposing units. When a break threshold is exceeded the unit is removed from action and neither causes attrition nor is attrited. Such a unit is not removed from the map display.

Ground Forces in ITEM are collections of ground units which move as a collective entity and maintain an internal structure. The units assigned to a ground force are named based on the force name from which they where derived. Figure 7 displays the attributes of the ground force. The name of the force uniquely identifies it. The location of the force, expressed in latitude and longitude, specifies the origin (0,0) of the axis system. The orientation of the force represents the direction perpendicular to the front of the force and defines the orientation of the axis on the map. This rectangle is re-oriented after force movements involving a change in direction. Attrition of the units of the force are degraded by the day and night weather factors as appropriate.

Air raids launched from both air bases and battle groups may attack ground forces. Ground forces do not currently interact directly with installations or naval objects.

C. EAGLE

First implemented in October 1992, Eagle was developed as an in-house project of TRAC in cooperation with Los Alamos National Labs (LANL) and the MITRE Corporation. Eagle is a two-sided, deterministic Corps/Division level combat model that simulates the land/air operational level of war and includes joint and combined operations. Human participation is limited to stopping the simulation and changing plans and orders for units. Eagle has been used to assess Courses of Action (COA), in decision support, as an exercise driver, to assess force composition, and as a staff trainer. The resolution is to battalion or company. Eagle incorporates object-oriented design and

implementation and was coded using both Common Lisp Object System (CLOS) and the Knowledge Engineering Environment (KEE) frame system. The Eagle object model discussed here will form the basis for a follow-on simulation, known as AWARS, which is currently under development. (TRAC, 1997)

Eagle is segmented into three distinct components formed from a set of knowledge bases. The knowledge bases separate objects based on functionality and permit the individual loading of only those knowledge bases applicable to a battlespace entity. Figure 10 gives one possible object model depicting the functionality of the military units in Eagle. Eagle contains a total of 31 knowledge bases supporting a total of 13,915 individual objects. All units and some portions of the simulation control mechanism are contained in the force-structure and characteristic knowledge bases. The actual unit and its functions reside in the force structure knowledge base while its assets and attributes are stored in the characteristic knowledge base.

Eagle requires a minimum of 715 data elements (including procedures) to describe a unit and thus has a very high level of resolution. A unit which has subordinate units that it must control to execute its mission has two sets of characteristics and is considered to be a tactical command post. This separates command units from combat units which are represented as resolution units. The tactical command unit inherits all of the same attributes as the combat unit but generally does not use them.

Command and control of Eagle units is accomplished through a series of attribute classes including planning, decision factors, battle operations, perceptions, and commo. These classes maintain the procedures for formulating battle plans taking into account the commanders battlespace awareness and the ability of the unit to communicate to other

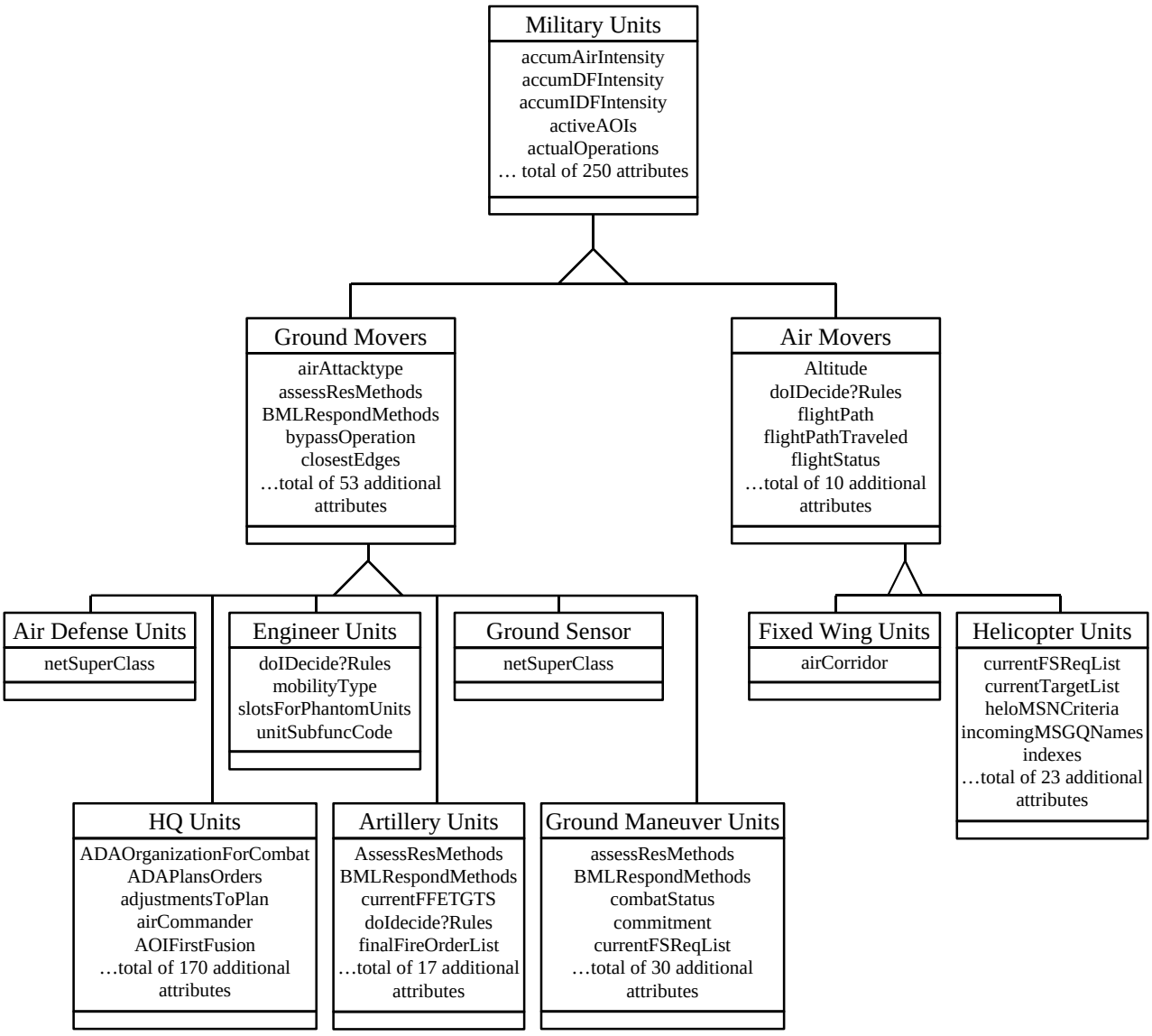


Figure 10: Eagle Class Hierarchy

units in order to achieve its assigned mission. Figure 11 shows possible object models for selected Eagle military unit command and control classes.

Unit movement is accomplished through its associated movement class. This movement is governed by a set of move rules which are based on a number of physical characteristics of the unit and environmental characteristics of the terrain. As depicted in

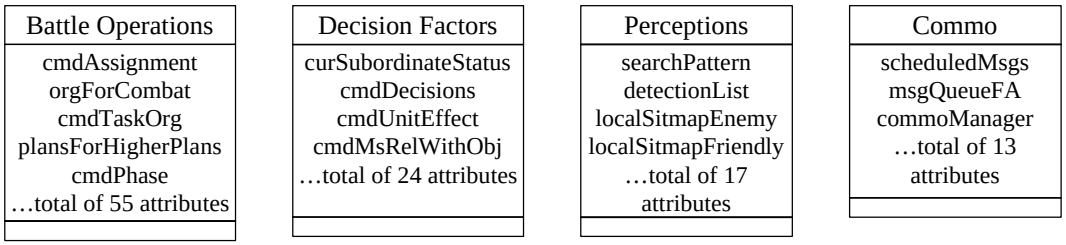


Figure 11: Selected Eagle Unit Command and Control Class Structures

Figure 12, the units grid location is stored as an attribute of the movement class.

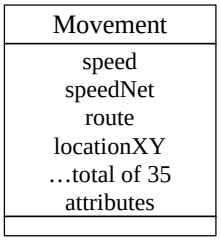


Figure 12: Eagle Movement Class Object Model

Unit attrition in Eagle is managed through the simulation control attrition knowledge base. Units decide who to shoot at and with what weaponry in their command and control functions. They than notify the attrition manager of all attrition pairings. The attrition manager resolves the attrition by examining the type of munitions and other characteristic data required to support the appropriate attrition algorithm. An example of the attrition manager class object model is presented in Figure 13. Eagle uses separate attrition algorithms for direct fire, indirect fire, minefield, etc.

By complying with HLA and drawing objects from the CMMS and FDB, AWARS, the follow-on to Eagle, will be quite robust and require little unique code or data.

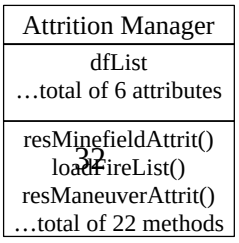


Figure 13: Eagle Attrition Manager Class Object Model

D. WARSIM 2000

Billed as the simulation which will train the Army's Force XXI commanders, WARSIM is under development by Lockheed Martin Federal Systems and Science Applications International Corporation (SAIC) and is scheduled for initial release in 1999. WARSIM will train these commanders in a realistic battlespace environment measuring human in the loop performance of tasks based on doctrine. HLA compliant, WARSIM will be an object-oriented, muli-sided, distributable system which will support the training of up to five echelons of command simultaneously. Further, the FDB will provide standard algorithms for unit and platform behavior providing a realistic battlespace. WARSIM interfaced with the Joint Simulation System (JSIMS) and the Combined Arms Tactical Trainers (CATT) will provide cutting edge training without the expense and risk to personnel and equipment of live exercises. (TRADOC 2, 1997)

WARSIM will portray all phases of Army combined arms operations in a land, air, and sea environment. These phases include mobilization, deployment, operations other than war, reconstitution, redeployment, and demobilization. WARSIM will model operations at levels from battalion through echelons above corps by aggregation of platforms and subordinate units. High-value, low-density systems can be modeled as independent systems. In the software requirements analysis phase, WARSIM units are presented using a condensed version of Rumbaugh's Object Model Technique (OMT).

Figure 14 depicts one possible WARSIM top level view of the unit Computer Software Configuration Item (CSCI) (Souder, 1997).

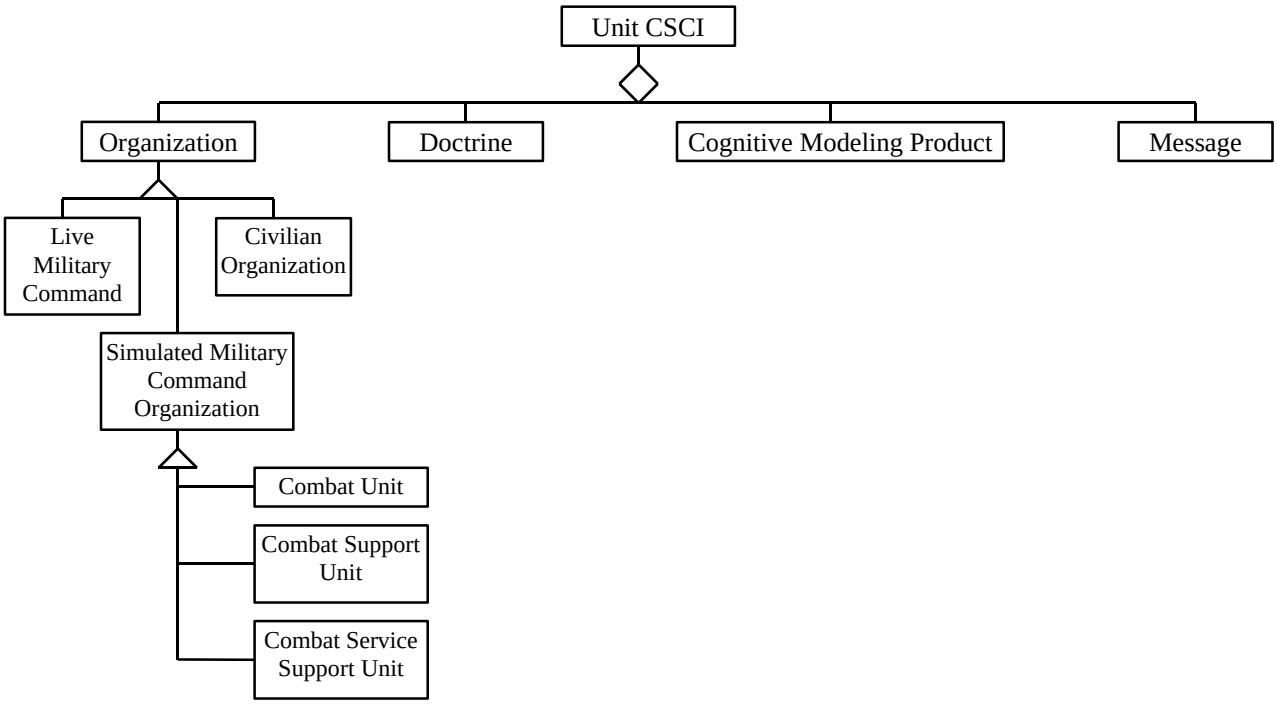


Figure 14: WARSIM 2000 Unit Top Level View (Souder, 1997)

In this figure, the unit is comprised of an organization structure, driven by doctrine, and able to communicate with other units. The cognitive modeling product represents the decision making process of the commander and staff. The class organization, captures the behavior of a group of people and the sub-class Simulated Military Command Organization adds the ability to direct other units and platforms. The Live Military Organization represents the human-in-the-loop interface in which a commander and staff are part of the training audience. Further specialization of the simulated military command organization includes the typing of units as either combat units (Field Artillery Units, Infantry Units, Aviation Units, etc.), combat support units

(Signal Unit, Military Intelligence Unit, Engineer Unit, etc.), and combat service support units (Medical Unit, Transportation Unit, Quartermaster Unit, etc.).

In Figure 14, the doctrine represents the data that provides the guidance and constraint for the execution of actions by the simulated organizations. Tactics, techniques, fundamental principles of war, and procedures are all types of doctrine. The cognitive modeling product is specialized into estimating the situation (perception of the tactical picture), planning (courses of action), and mission (control measures and detailed execution matrixes of assigned tasks). Figure 15 is a visual depiction of the associations between units. The message class permits the passing of command, control, and intelligence information among units. Figure 15 shows non-tangible elements of a commander's decision making process including a plan object class which is capable of considering several courses of action that could successfully accomplish a mission. This

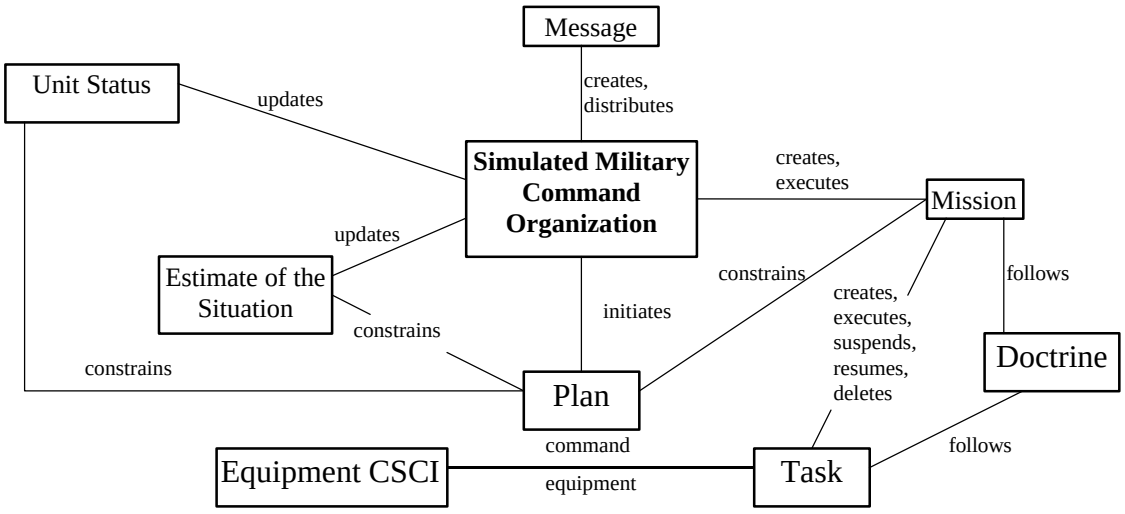


Figure 15: WARSIM 2000 Unit Object Associations (Souder, 1997)

approach allows WARSIM to express and interpret the commander and staff cognitive process to capture the users intentions and requirements.

While not fully developed yet, unit movement and attrition in WARSIM are expected to be similar to ModSAF in that units will act as an aggregation of its components. This approach is well suited to the detail of the model, the use of the Conceptual Model of the Mission Space (CMMS) for modeling environmental entities, and the Functional Description of the Battlespace (FDB) for descriptions and performance parameters of battlespace entities.

E. JWARS

JWARS is being developed by the Office of the Secretary of Defense (OSD) as the next generation of the Tactical Warfare (TACWAR) model. A significant component of the Joint Analytical Model Improvement Program (JAMIP), JWARS will be a state-of-the-art, object-oriented, closed form, constructive simulation of multi-sided, joint warfare for analysis. The principal users of JWARS include Combatant Commanders, Joint Staffs, Service Staffs, OSD, and other DoD organizations. (JWARS, 1996)

JWARS will be developed incrementally over three blocks. The JWARS prototype has been implemented and is currently under assessment while the initial operational capability version, Block I, is scheduled for release in December 1998. The objectives and scope of JWARS are quite comprehensive. Command, control, communications, computers, intelligence, surveillance, and reconnaissance (C4ISR) will serve as the foundation for the model. The effects of the physical environment (terrain, ocean, air, and space) on the simulated activities will be modeled. Most interestingly, the model is required to be sufficiently flexible to deal with future warfare concepts, doctrine, systems, and organizations not only for the United States but also for both its potential allies and potential foes. To accomplish this feat, JWARS refines the modeling of

collective planning, threat situation development, and dissemination of intelligence within the mission space. It also includes Course of Action, commander's assessment, and targeting. The resolution of JWARS includes ground forces to battalion level (maneuver) and battery (air defense), naval and air forces to combatant (ship and flight) level, and theater and national sensors and precision strike weapons at the system level.

The overall schema of JWARS is captured in its High-Level Object Model. Figure 16 diagrams the major object classes and their associations identified through analysis of the JWARS problem domain. The Command class is the legal and authoritative leadership organization of the force. The C2 Element class represents the staff planning and thinking

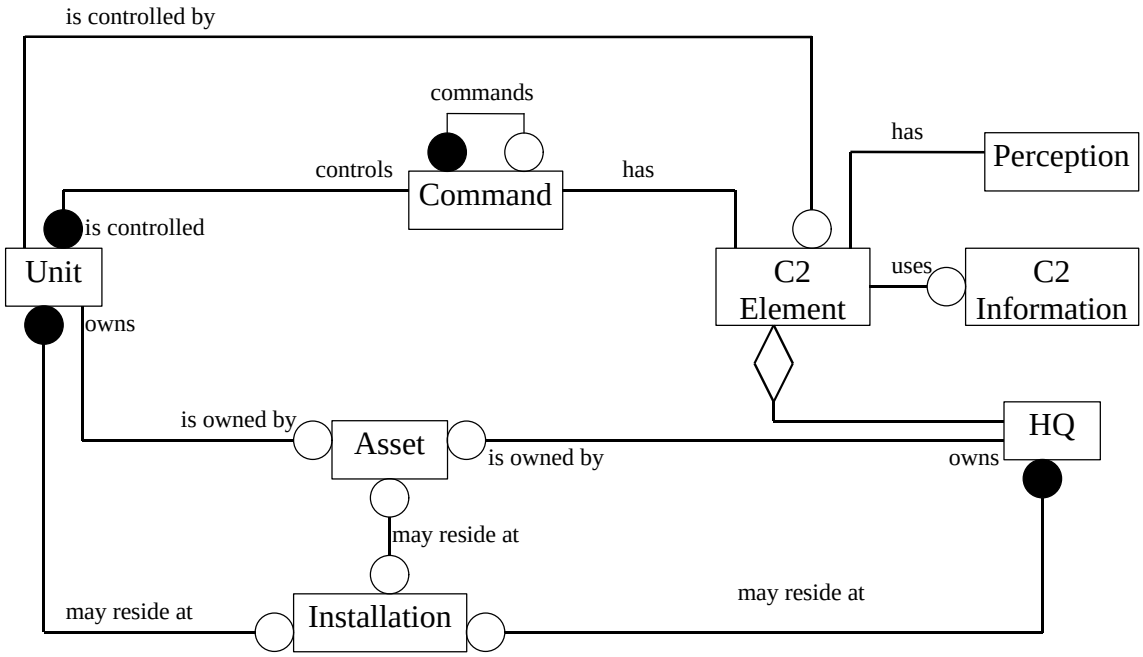


Figure 16: JWARS Top-Level Object Model (JWARS, 1996)

capability of the force. The Unit class is analogous to the body of the force that is capable of executing missions. The Asset class models the tools. used by the units to perform its mission. The Installation class represents facilities which support particular functions.

Figure 16 demonstrates JWAR's emphasis on command functionality. Perhaps the most interesting feature of JWARS is its C2 Element Class shown in Figure 17. The C2 Element class, representing the generic functionality of all commands and units, may be either a command or a unit element. The nationality defines the country to which the C2 Element belongs. Side characterizes a C2 Element's role in the scope (representing friendly, enemy, neutral, and coalition forces) of the operation. The area of responsibility (AOR) designates the physical area over which the C2 Element is

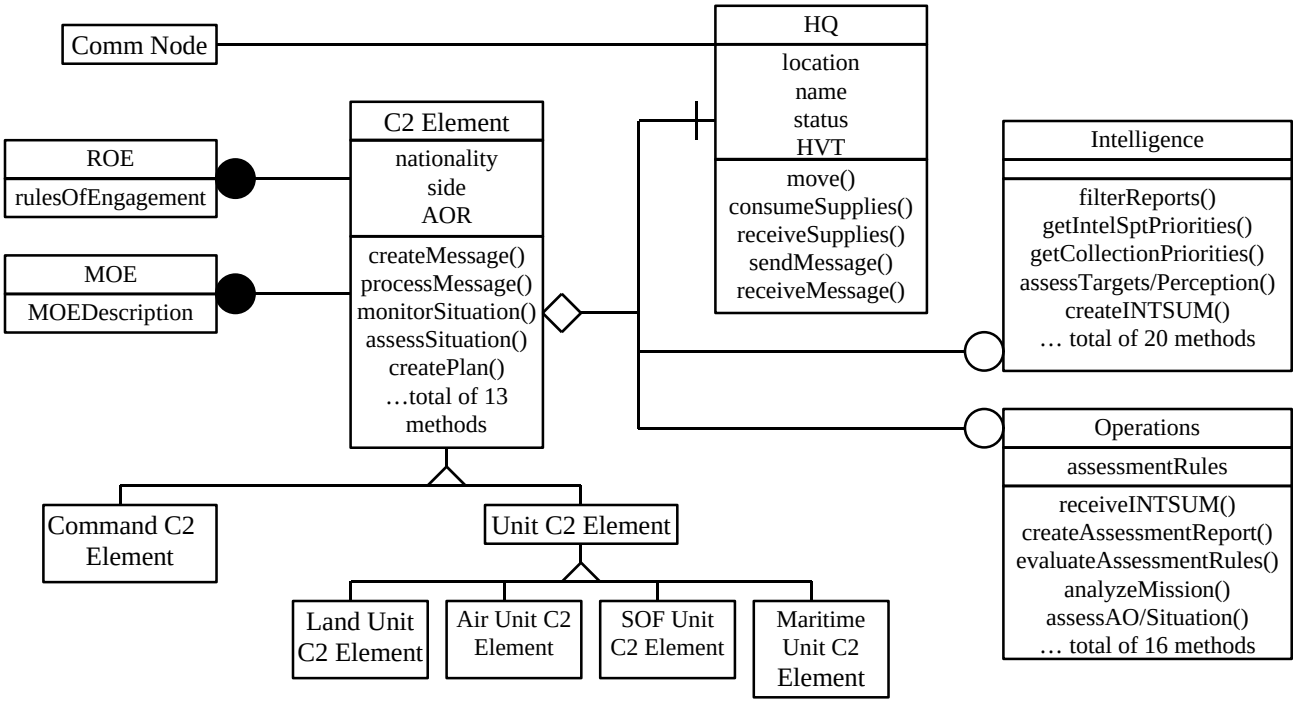


Figure17: JWARS C2 Element Class (JWARS, 1996)

responsible. Each C2 Element class is composed of one or more HQ classes and may be composed of an operations class and an intelligence class. Not specifically oriented

towards any command or unit entity, the C2 Element class is intended to encompass broad generic behaviors of all C2 elements. The HQ class is used to represent the physical attributes and methods associated with a C2 Element. The HQ moves, consumes and receives supplies, and sends and receives messages. The Operations class performs the situation development activities for a C2 Element using rules of engagement (ROE) and measures of effectiveness (MOE). During situation development the mission is analyzed along with enemy capabilities to create a variety of courses of actions (COA). The Intelligence class performs the collection management activities for a C2 Element.

The Unit class, shown in Figure 18, is used to represent battlespace entities which carry out actions in support of a mission. A Unit's side represents its status as friendly, enemy, neutral, or coalition. The Table of Organization and Equipment (TOE) identifies the assets required for the formation of specific units. A Unit moves from one location to another either on foot or aboard some type of platform. Units execute orders received from their assigned headquarters. Whether or not the Unit is currently involved in an ongoing mission, supplies are consumed and received.

The Unit Class is to be used for aggregate representations as distinct from the Asset Class which will be used for singular representations. The aggregate representations are also referred to as resolution units. For the Army this includes brigade battalion, or company. The resolution unit is the smallest organization that will be instantiated for the particular study. These resolution units will require adjudication algorithms that are matched to the level of aggregation of the Unit class instances participating in an interaction.

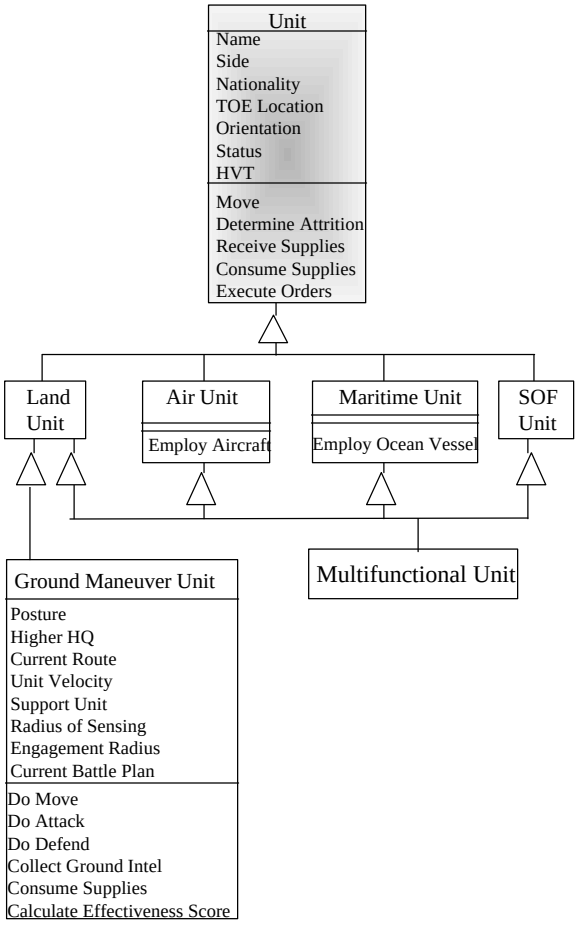


Figure 18: JWARS Unit Class (JWARS, 1996)

The Ground Maneuver Unit is used to model Army forces in JWARS. A Ground Maneuver Unit possesses a Battle Plan which is modified by fragmentary orders from higher headquarters. These orders alter the posture of the Unit as it moves toward the objective specified in the plan.

The minimum requirements of both legacy and developmental model units incorporate the ability of a unit to command and control other units and platforms, have some method of movement, and can cause and receive attrition. These elements will form the basis of the proposed standard unit level object model.

IV. PROPOSED MODEL

In order to promote design flexibility, component-based modeling is used. The primary standards are specified for components of entities rather than the entities themselves. The ability to assemble the components into entities in different ways simultaneously increases both design flexibility and object reuse. Generally, associations between components and classes are made possible by polymorphism. Polymorphism permits substitution of compatible components in an entity. For example, a Field Artillery Battalion's maintenance capabilities may be improved to represent the arrival of more maintenance assets by simply replacing the appropriate component.

A. LEVEL OF DETAIL

The component-based design of standards is intended to be independent of code, and therefore make no specification or restriction as to how its classes, methods, and objects are to be implemented. In fact, the only supposition made is that the implementation language is able to support object-oriented programming. Specifically, the implementation language must support inheritance, polymorphism, encapsulation, abstraction, and overriding as discussed in Chapter I. The proposed standard abstract unit class is designed to be subclassed during implementation to achieve the desired level of specialization. This class is not meant to be comprehensively detailed and, in fact, is deliberately minimal by design. To illustrate this idea Figure 19 has broken the spectrum of models into four artificial levels of completeness ranging from components (Level 0),

abstract classes (Levels 1 & 2) to highly detailed, instantiable objects (Level 3). Clearly, flexibility and reuse are greatest at the lower levels. The proposed Unit class lies near

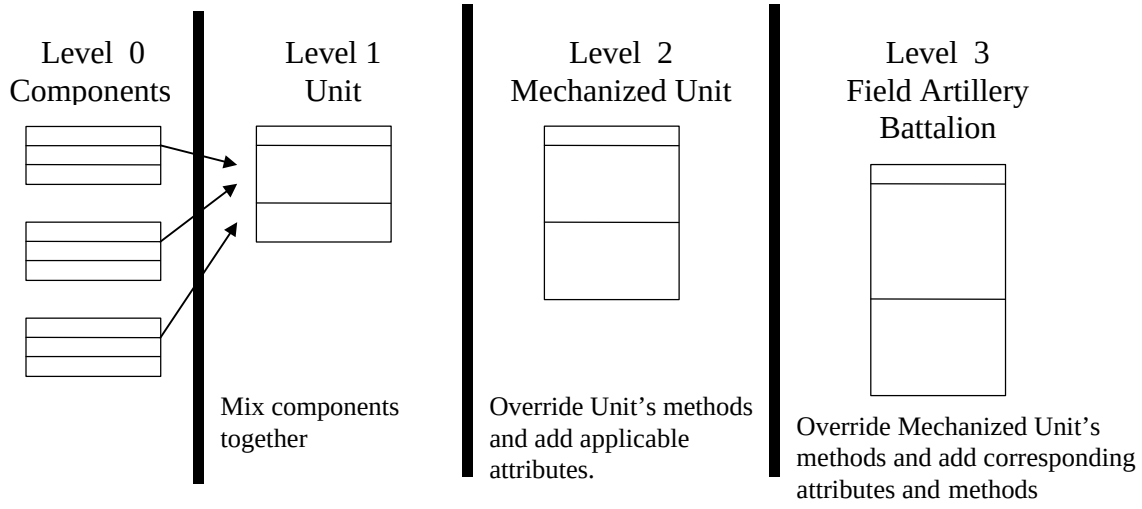


Figure 19: Level of Detail (Jackson, 1997)

Level 1 is composed of components from Level 0. A Ground Maneuver Unit may be considered closer to Level 2. Support for the proposed class will be provided by enumerating all of the units at the second level and offering possible implementation examples which are at the third level.

B. UNIT COMPONENT HIERARCHY

The Unit class is composed of one or more Unit Components. Unit Components may be viewed as the building blocks for the force structure in the simulation model. Each Unit Component is responsible for a specific set of related tasks in support of the unit's mission. They are designed to be sufficiently generic that most existing simulation models could easily map their functionality into the Unit Component structure. The top level of the class hierarchy of Unit Components is shown in Figure 20.

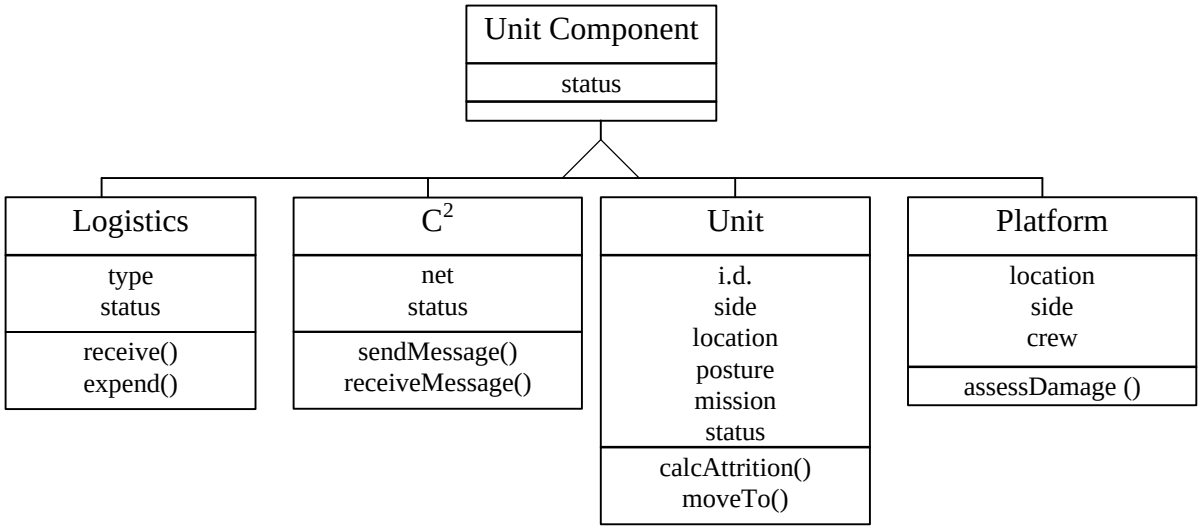


Figure 20: The Unit Component Class Hierarchy

The root of the tree is the Unit Component class, which has a single attribute, status. The status attribute is meant to describe the degree of functionality of a Unit Component. The simplest possible implementation amounts to a Boolean which indicates whether the component is functioning or not. A more complex approach could include a percent effectiveness or even multidimensional variables. Of all the attributes and behaviors of unit components, its status was the only common feature. Placing the status attribute in the root of the class hierarchy allows a component of any type to be queried about its status without having to know the precise class to which the queried component belongs.

There are four immediate descendants of the Unit Component class: Logistics, Command and Control (C^2), subordinate Units, and Platforms. This is intended to be a comprehensive set of classification types of components used to create units. Each is endowed with only those attributes and methods necessary to specify its generic behavior. Each is an abstract class because they represent a conceptual functionality of the unit rather than a concrete entity.

1. The Logistics Class Hierarchy

The logistics class is intended to capture attributes and methods common to all logistics units and is subclassed for refinement into two classes: maintenance and supply, depicted in Figure 21. The supply class may be specialized according to its type by overriding its methods. For example, the supply component may be a Food Service Section, an Ammo Section, or simply an aggregated placeholder for these logistics functions supporting the desired Unit structure. This design allows the modeler to fully incorporate logistics structures into combat models while permitting more realistic modeling of sustained campaigns. Conversely, since

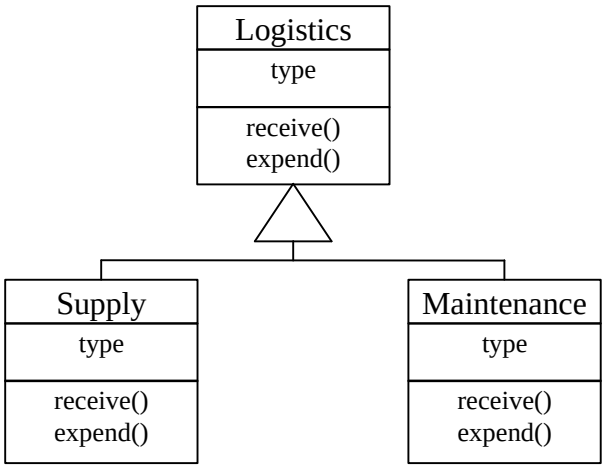


Figure 21: Logistics Class Hierarchy

logistics is not a mandatory subclass, it may be omitted if not required for the purpose of the simulation. The maintenance class represents the organic maintenance capability of the associated unit. The type attribute denotes the logistic component's mission, for example, Class III, Class V, or aviation maintenance. The behavior receive() is used to model the receipt of supplies or entities to be repaired, while expend() marks the consumption of material or return of a repaired asset.

Observe that the primary difference between the immediate subclasses is their respective names. Each simply adds a different override to the methods receive() and expend(), so the underlying method algorithms are appropriate for the mission.

2. The Command and Control Class

Issuing and receiving orders and other communications between units are conducted via the component object Command and Control (C^2). The C^2 class, abstracting all forms of communication, ensures that all units are guaranteed the essential ability to communicate with other units. As will be illustrated later, this is the only mandatory component of a Unit. The precise characteristics of the instantiated class would depend on the kind of communication involved (captured in the attribute labeled net). Figure 22 depicts the C^2 class.

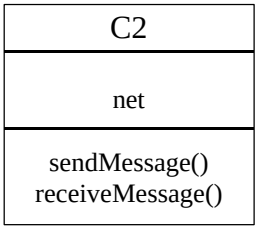


Figure 22: C2 Class

One possible use of the C^2 class would involve incoming orders to be captured by the receiveMessage() method and then posted to the event list for the appropriate component.

3. Subordinate Unit Component

The unit component is used to show force structure by developing senior/subordinate relationships between the individual strategical/tactical units. The unit class is discussed in detail in Section C. A unit may control none or many subordinate

units. This design provides the modeler a great deal of flexibility in organizing and analyzing force structure.

4. Platform Component

The platform component of a unit is the platform class discussed in Captain Doug Dudgeon's thesis. (Dudgeon, 1997) Platforms are considered to be concrete objects which have the innate ability to carry weapons or perform tactically important military functions. Like the Unit level class, the standard Platform is also an aggregation of its components. This design marks the interface between a tactical unit and its equipment. The unit may have none or many platform objects. The platform component may be discriminantly used to analyze a high interest system such as a patriot battery against an enemy using SCUD missiles. There are eight immediate descendants of the Platform Component class: Sensor, Weapon, Movement, Supply, Communication, Carrier, Hull, and Platform (Figure 23).

The Sensor, Weapon, and Movement classes are used to capture the basic "look," "shoot," and "move" categories. The Supply class represents things that are consumed by platforms. Communication between other platforms and units is modeled in the Communication class. Propulsion of platforms is modeled in the Movement class. The Hull class contains the physical or performance specifications of the platform. The Carrier class models the platform's capability to carry other platforms such as an Infantry Fighting Vehicle transporting class models the platform's capability to carry other platforms such as an Infantry Fighting Vehicle transporting a squad of infantry. The components depicted

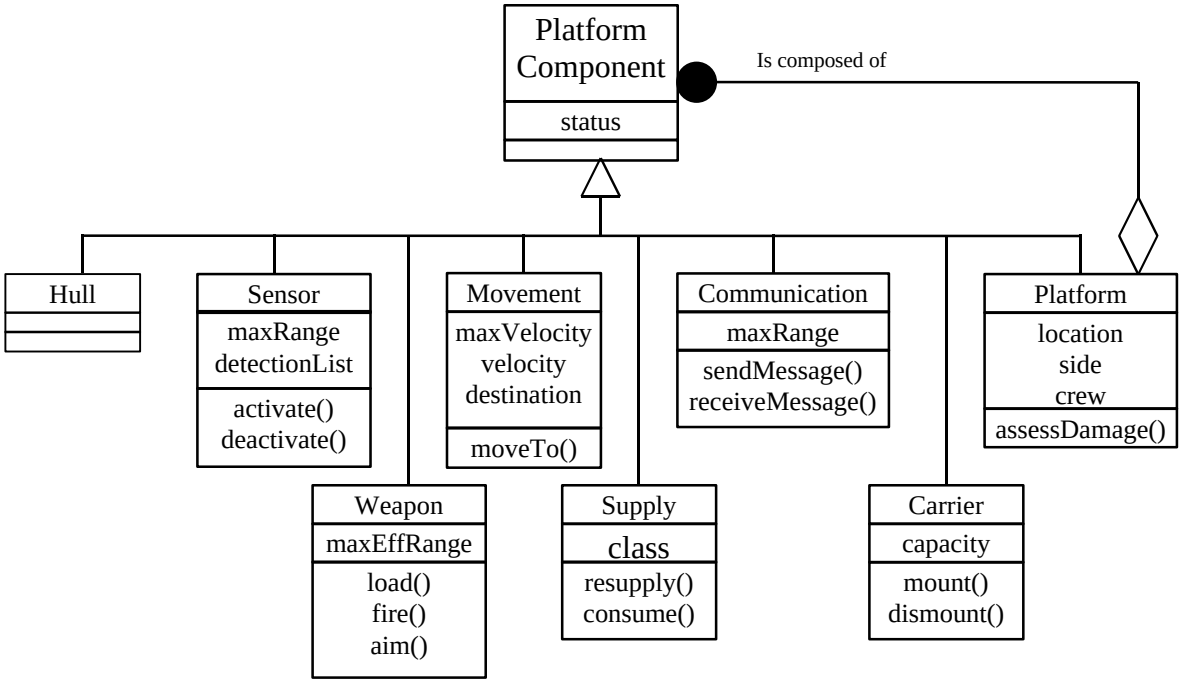


Figure 23: Platform Component Class Hierarchy (Dudgeon, 1997)

above are aggregated into the basic Platform class, as shown in Figure 24. Note that Figure 24 displays associations rather than a class hierarchy.

The Platform class adds attributes of location, side, and crew as well as a assessDamage() method. The associations in Figure 24 indicate that every platform can

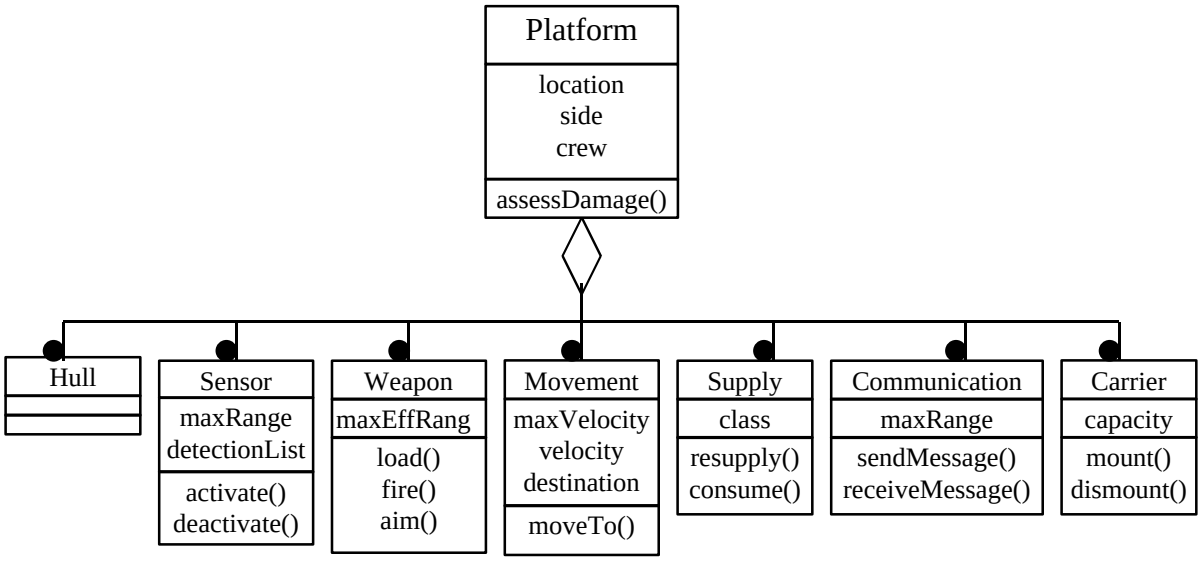


Figure 24: Platform Class Associations (Dudgeon, 1997)

have none or more of each of the particular platform components. The assessDamage() method is responsible for specifying the Platform's behavior when it is hit. All other properties and behaviors are delegated to the various components of the Platform. The Platform delegates most of its functionality to its components, and the methods invoked are generic. (Dudgeon, 1997)

C. THE UNIT CLASS

The Unit class is used to represent battlespace entities which direct its components to carry out actions in support of a mission. The unit components discussed previously, are aggregated into a basic Unit class, as shown in Figure 25. The associations in Figure 25 indicate that every unit must have at least one command and control component and can have none, one, or more of each of the remaining Unit Components. This gives considerable flexibility to the modeler while still enabling substantial reuse.

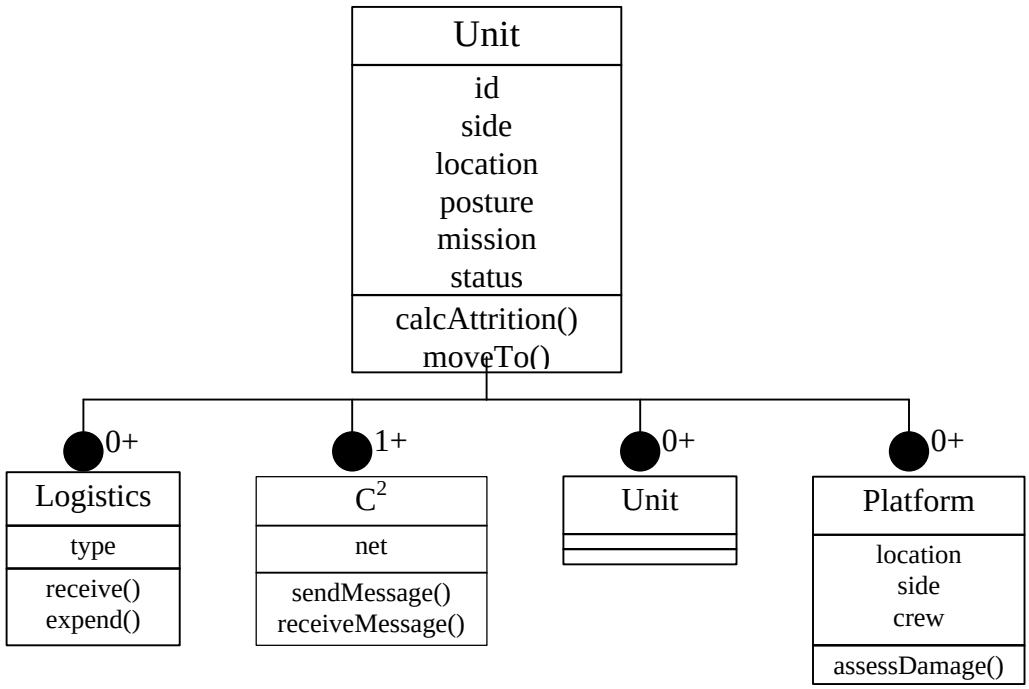


Figure25: The Unit Class (Aggregation)

The Unit class adds the attributes id, side, location, posture, and mission to the component attribute labeled status. Also added were the methods calcAttrition() and moveTo().

1. Unit Attributes

a) id

The attribute id is a label for specifying the name or place (a hierarchical position) in the force structure of the instantiated unit. The modeler is free to implement this attribute as best suited to the simulation. A possible approach is to use the Unit Identification Code (UIC) which uniquely identifies each active, reserve, and National Guard Unit of the Armed Forces. An alternate implementation may be to list the unit commander in the id field to aid in simulation analysis. This attribute may even be used to classify units for graphical display purposes.

b) side

The attribute side is a label indicating which forces are fighting together against another potentially multi-sided force. Implementation of this attribute could be to use colors, nationalities, or coalitions. This attribute is common to all models investigated, regardless of whether the simulation is only two-sided or multi-sided.

c) Location

One of the most fundamental properties of all entities in a simulation model is that of its location in the simulated coordinate system. The class hierarchy is shown in Figure 26. The location attribute is a class defined to specify (abstract) methods for computing the distance from another location (distanceFrom()) and methods

converting from one type of coordinate system to another (convert()). The attribute orientation represents the direction the unit is facing. This attribute is used to determine limits of component sensing

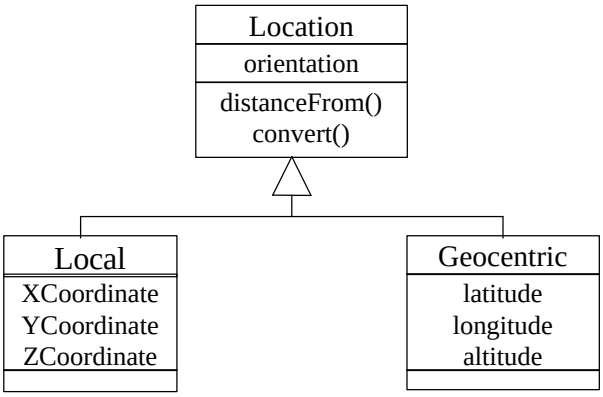


Figure 26: Location Class Hierarchy

and possibly attrition of other units. The methods defined in the superclass can be utilized by any other instance of location.

The subclasses of Location are Local and Geocentric, representing two fundamental ways of representing locations. There are many different types of coordinate systems, each suitable for some uses and not for others. The standards enable extensibility to other coordinate systems by stipulating one base coordinate system that every other coordinate system must provide conversion to. The proposed base coordinate system is the Cartesian coordinate system.

Location is a fundamental property of all entities in a simulation, and all models implement some form of it. A primary benefit of a Location hierarchy rooted in an abstract Location class is that each model is free to use the approach that is best suited to its domain or the one that is standard within its community. For example, models that depict ground warfare typically use local coordinate systems, whereas environmental

models use geocentric coordinates. With the Location hierarchy, the fundamental algorithms would not have to be changed, yet a certain degree of interoperability between models would be established. Models that internally represent their location data in different coordinate systems would be able to have their locations consistently represented in the other. Thus, both interoperability as well as reuse is achieved.

d) posture

The Posture is a label depicting the units actual present course of action. This may include the task for carrying out of strategic, tactical, service, training, or administrative missions. The list of possible postures might include; attack, defend, retreat, hasty retreat, awaiting further orders, search, and relocate. Once again, this list in not meant to be exhaustive, but rather exemplary of possible postures the unit may assume. This list will be dynamic and will be changed as doctrinal nomenclature changes in response to revolution in military affairs. The posture of the unit is a determinant in potential employment, movement, and attrition.

e) mission

Several possible implementation schemes exist for the attribute mission. Mission is considered to be a task combined with a purpose which clearly defines an ultimate objective and the reason for attaining that objective. One possible implementation, then, is to have the attribute represent a list of taskings assigned by higher authority. These tasks, in turn, are comprised of processes that describe how functions are to be performed. These tasks might be doctrinally based and may, at times, be reactive in nature, as in the case of contingency plans. An alternative implementation of the attribute mission is to use it as a label holding the phase of the unit's current

assigned mission. In this form the attribute would update as a unit completed enabling tasks defining different phases of a mission. Taken together with the unit's posture, its mission could impact the appropriate selection of algorithms for movement, communication, and attrition.

2. Methods

a) calcAttrition()

The calcAttrition() method calculates the losses to a unit resulting from armed conflict. Various standard attrition algorithms could be appropriately applied depending on the unit's posture and mission. Attrition could be either to the opposing unit or to the unit invoking the method. This method of applying attrition algorithms permits the modeler to analyze the effects of using different algorithms.

Another consideration for the ultimate implementation of this method is the intended resolution of the model. This method can accommodate a high resolution model which treats individual platforms with attrition calculated as a firing weapon against an intended target. Alternatively, an aggregate resolution model may implement this method as units (e.g., battalions, brigades, etc.) attriting other units. In this case, the unit might occupy an area or take on some geometric battlespace and its lethality is an aggregate of its subordinate components.

b) moveTo()

The moveTo() method starts a unit moving according to the unit's platforms particular movement algorithm. Various standard movement algorithms may be used for the implementation of this method. The rate of movement of a unit may be considered to be an average of its components' rates of movement or may be limited to

the rate of its slowest component. The implementation will depend upon the purpose and resolution of the model. Higher resolution models typically allow individual platforms to move independently or in some formation, while the unit location is considered to correspond to some headquarters or other meaningful position. The unit's posture and its surrounding environment and terrain will also factor into the rate of movement. Additionally, the modeler may add some restrictions due to maintenance and petroleum, oil, and lubricant (POL) availability. Again, the modeler is free to determine if the unit's location is reported as the forward most unit or the location of the headquarters unit, or any other meaningful position.

D. THE UNIT CLASS HIERARCHY

In order to give flexibility to the modelers and developers of simulations and to avoid dictating implementations as much as possible, the focus is returned to class hierarchies. Of the many possible hierarchies, Figure 27 depicts the minimal level of specifications consistent with flexibility and reuse. Recall the discussion presented in section A, particularly Figure 19. Figure 27 starts at Level 1 with the basic Unit and

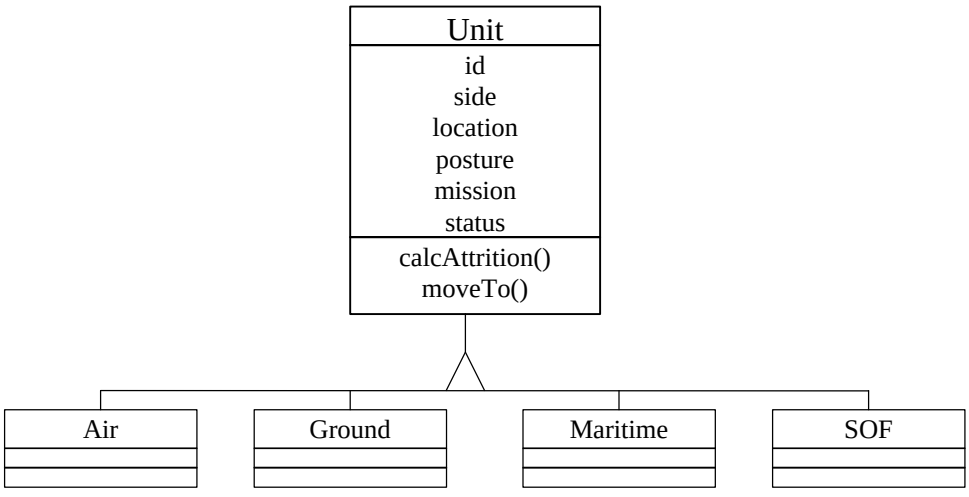


Figure 27: Unit Class Hierarchy

moves through Level 1.5 by showing the Air unit, Ground unit, Maritime unit, and Special Operations Forces (SOF) units.

The Multifunctional Unit class highlights an interesting implication for multiple inheritance. Multiple inheritance offers the flexibility and potential reuse of classes in order to add functionality safely which is not possible using only single inheritance. The other method which holds this promise is single inheritance with multiple interfaces.

1. Multiple Inheritance

Ordinary multiple inheritance is when a class can inherit attributes and methods from more than one superclass. This ability immediately creates many possibilities for robust reuse of classes. To illustrate, consider modeling a unified command staff in a high level-low resolution simulation. This staff would inherit attributes and methods from unit classes modeling all battlespace domains. An instance of a unified command staff (USCENTCOM for example) would be an Air unit, a Ground unit, a Maritime unit, and a SOF unit. Since it would contain all the attributes and could respond to all the methods of all of its superclasses, objects interacting with the unified command could perceive it accordingly. The superclass is reused because the subclass does not have to re-implement the inherited methods and attributes.

This design is not without potential pitfalls. If two superclasses have the same method with the same signature, then it is not possible to determine which, if any, of the methods is to be used when the new class invokes it. Additionally, different orders of execution could have radically different and unpredictable results. This situation violates

encapsulation in that the details of each superclass constructor must be known by the subclass in order to properly resolve the order.

2. Single Inheritance With Multiple Interfaces

The second approach, single inheritance with multiple interfaces, avoids most of the difficulties of unrestricted multiple inheritance. This design permits only one superclass to be concrete (instantiable) while the remainder are purely abstract. These abstract superclasses contain only functions which effectively form a cluster of behaviors among the superclasses. Since the subclass has at most one inherited version of a method, there is no ambiguity of inherited methods.

The Unit delegates much of its functionality to its components, and the methods invoked are generic. Reuse is achieved through this generality. The classes of the components (logistics, C^2, unit, and platform) are reused by virtue of the same component class being instantiated for possibly many different Units. The component class need not be rewritten or even recompiled. Extensibility is achieved by subclassing existing component classes. For example, if a new force structure were proposed it could be brought into existing models by subclassing existing Unit classes. The new force structure could be associated with an existing Unit without having to modify the Unit in any way. Furthermore, the new Unit can interoperate with all simulations it could previously.

An example of how Unit can be usefully subclassed is the Field Artillery Battalion (FAB) class, shown in Figure 28. In the FAB class, concrete subclasses of the Unit Component classes are inserted to perform the various functions. Certain of the associations are made more specific as well. For example, while the Unit class specifies

zero or more Logistics, The FAB specifies zero logistic units. Instead the logistics functions of the FAB are subclassed to the Service Battery (Figure 29). Likewise, no Platforms belong to the FAB but rather they reside within the HHB and FA Battery units.

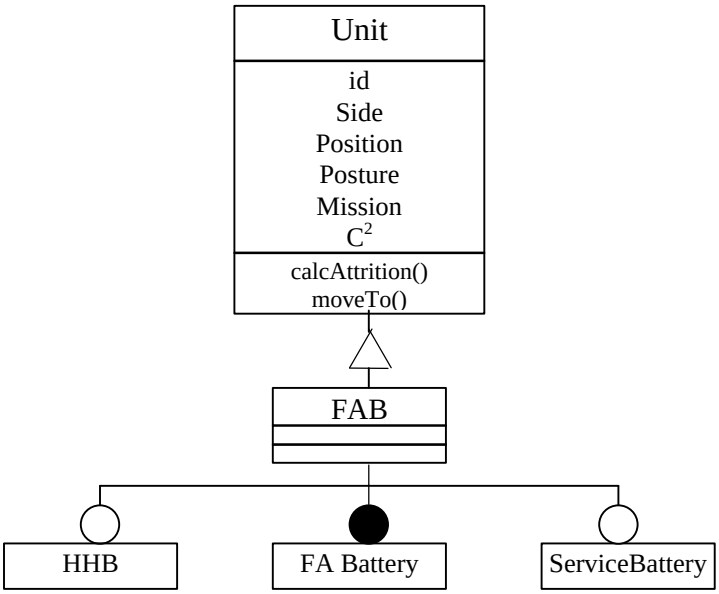


Figure 28: The Field Artillery Battalion (FAB)

There are three sub-ordinate units associated with the FAB, namely the Service Battery,

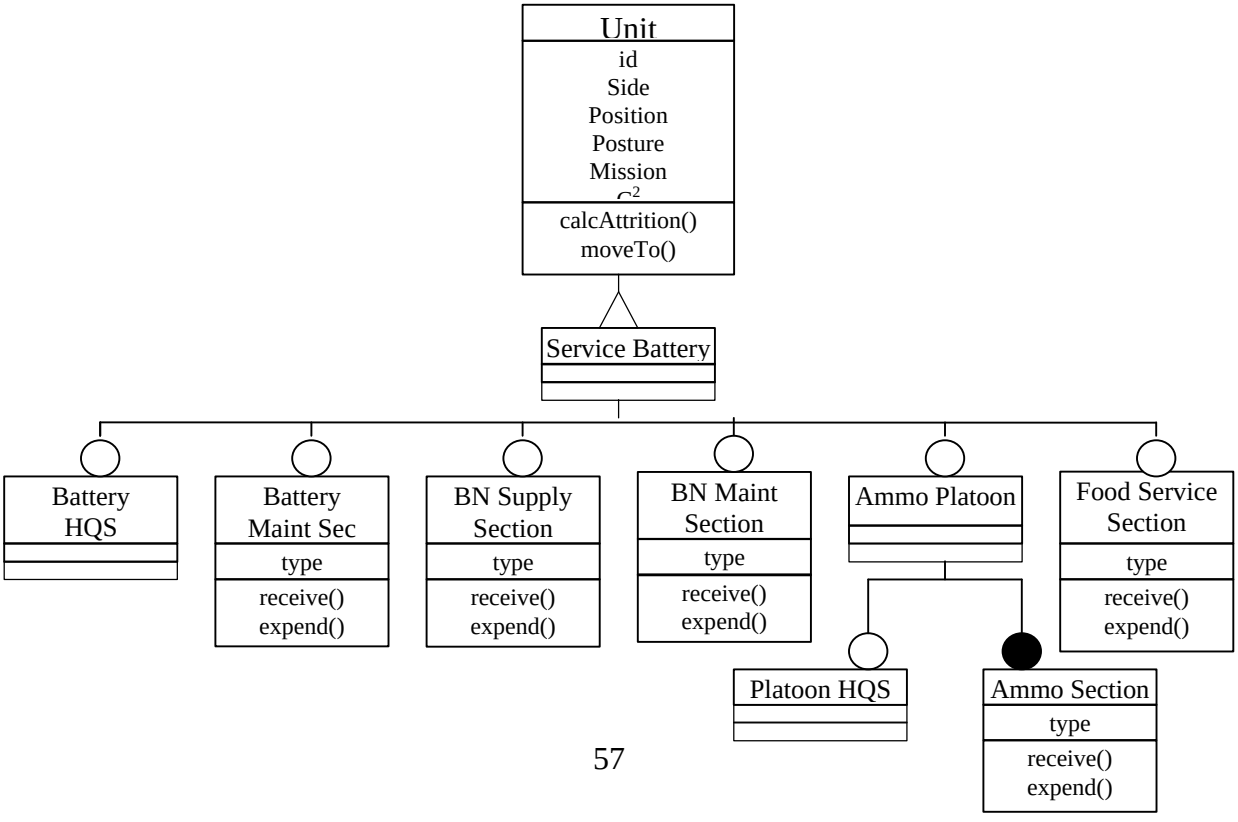


Figure 29: The Service Battery Class

HHB, and FA Battery. Recall, the Unit class specifies zero or more subordinate units. In this example we only roll one layer into the FAB but we know how many types of units are ultimately associated by the labeling convention of the id attribute.

The Field Artillery Battalion and Service Battery are just two examples of possible mapping of the standard unit-level object model (level 1) to concrete and instantiable objects (level 2). Specifying a unique set of data and behaviors for these objects would complete the transition to level 3 objects. Similarly, the standard unit-level object model must provide the framework for the conceptual mapping of entity attributes and behaviors in legacy and future simulations to an object representation.

E. COMPARISON TO LEGACY AND DEVELOPMENTAL MODELS

An initial projection of the minimum essential packaging of the standard object can be derived from those features common among legacy and developmental models. Essential is not meant to imply that an object could not be implemented without the feature, but rather that the feature captures an element which inherently forms a basis for a combat unit.

1. Attributes

The data structures selected for inclusion in the proposed model map well with all models examined for this thesis. All models have a unique way to identify the unit whether it is for graphical or analytical purposes. As an example, ModSAF has an attribute named “callSign” and ITEM and JWARS use the attribute “name” to hold the

unit's identifier. Similarly, all of the models used an attribute to mark the division of the forces into sides. ModSAF is one of the models which uses colors as data elements for the attribute side. JWARS uses the actual nationality of the unit and can form coalitions based on the this attribute. All of the studied simulations had attributes for location. To capture the concept of the attribute posture, ITEM associates a posture class with a class called pathPoints. The pathPoints are based on the current mission of the unit and include tactical orders for the unit as it progresses towards its ultimate objective. ModSAF incorporates posture into an associated class of Rules Of Engagement (ROE). In Eagle, portions of the posture function are found in both the perceptions and battle operations classes. JWARS also has a posture attribute. WARSIM and all of the other models studied associated a mission to each unit. Additionally, ITEM and JWARS keep a log of assigned missions as they change over the course of the engagement.

2. Methods

The behaviors selected for inclusion in the proposed model also map well into all models examined for this thesis. The ability of the unit to move and its ability to realistically cause attrition and to be attrited are minimum essential behaviors of military units. Eagle has the most unique implementation to effect attrition. Using an omnipotent class called Attrition Manager, Eagle controls all attrition during an engagement in one class rather than having individual units call the algorithms. The most fundamental element differentiating a unit from a platform is the unit's ability to perform command and control functions. The proposed model unit component class C^2 could be implemented to hold all forms of command functions and could be specified in the implementation to permit any degree of human intervention. This corresponds to the

various association of classes used in the legacy and developmental models to perform these functions. The purpose and resolution of the model dictate the final structure of the command and control function. While ModSAF does not incorporate logistic functions in its model, all other simulations had, as a minimum, the capability to receive and expend supplies.

All of the models analyzed use a modular approach in developing their object models, but each uses somewhat different nomenclature to represent identical objects. The proposed standard unit-level object model will at least, provide a common language for the development of new simulations. Using standard algorithms will also ensure that appropriate behaviors are associated with this common nomenclature.

F. STANDARD ALGORITHMS

Similar to the arguments made for standard objects in Chapter I, standard algorithms also offer the possibility of reuse and interoperability. In a parallel study to the standard object study, the US Army Modeling and Simulation Office is leading an effort to establish standard algorithms for several categories: terrain, target acquisition, mobility, attrition, reasoning, supplying, servicing, and communications (AMSAA, 1996). All of the developed standard algorithms will conform to the set of developed standard objects. This association of standard algorithms to standard objects is synergistic.

Regardless of the class of standard algorithm, each standard algorithm requires a partitioned data set. Data will be both passed by the sending object through the signature of the method and provided by the receiving object. The specific signature and repository for the required data is left to the model developer. Consider the elementary form of the

Lanchester Linear Law which requires an attrition coefficient (in units of casualties/(time*number of firers*number of targets)), a time increment, and respective force levels of the combatants. The data are naturally partitioned so that the sending object provides the number of firers and perhaps the time increment of the engagement. Likewise, the receiving object must provide the number of targets and a calculation of the attrition coefficient. The standard attrition algorithm, then, implies that the calcAttrition() signature must contain the number of firers, a time increment, and a unit positional aspect or other pertinent data necessary for the calculation of the attrition coefficient.

Direct fire attrition in aggregate resolution combat models is often depicted as the aggregate kill rates of the attacking units weapons systems versus the engaged unit's targets. The kill rate might be a function of a number of performance and environmental parameters including: the number of functioning weapons by type, their lethality against different types of targets, rates of fire, ammunition supply, and terrain and weather factors. In contrast, area fire in aggregate resolution combat models is represented a methodology which generates an aggregated probability of kill for each firing mission against each target type in the engaged unit. In this case, the impact point and individual damage effect of each munition is not calculated against each individual target platform.

G. DISCUSSION

The standard unit-level object model and its components were based on the core competencies of military units: planning, communicating, command and control, shooting, movement, and sustainment. This logical division of essential functionalities is

shared by the object models which were studied. Thus, the standard unit-level object model can serve as a bridge between legacy and developmental simulation models.

Since the standard unit-level object model is designed to be independent of implementation, it does not specifically dictate associations. As a result of gaining this flexibility, interoperability may be slightly reduced. As standard algorithms and data structures are developed some associations may be outlined in the standard unit-level object model. In the interim, the standard unit-level object model could be used as a tool to display and evaluate the interactions between the recommended standard data and algorithms.

V. CONCLUSIONS AND RECOMMENDATIONS

Analytical models the size and complexity of projected future Army simulations require extensive front-end analysis, such as that conducted in this thesis. This analysis has researched the characteristics, behaviors, and interactions among many possible battlespace entities. Every effort was made to understand and describe the functions, processes, and tasks from the "real world" that may be represented in the model.

A. SUMMARY

The research and experience culminating in this thesis support the development and continuous improvement of the standard unit-level object model. This effort will greatly reduce the fragmentation found in legacy combat simulations. All of the simulations studied had very similar object models whose functionality could be built by appropriate implementation of the standard unit-level object model.

Initial attempts to define standard units led to classifying units based on mission type. This led to separate unit classes for all service branches having subclassed units including headquarters, communication, air defense, aviation, and maintenance. Further examination proved that this classification could be generalized to combat and non-combat classed units. To maximize flexibility and extensibility, the single standard unit-level object model formed from any number of standard components was proposed.

The methods contained in the standard unit-level object model allow communication between a unit and its components. Placing these methods in the standard unit-level object model maximizes the benefits of polymorphism and allows other objects to access these methods without prior knowledge of the specific class of the

object. Interoperability is achieved by the specification of this minimal set of methods which provide a ready interface to other simulations. Simulations designed around the standard unit-level object model and the standard platform-level object model will not only be able to interact with similarly designed models, but they can also reuse object models and code. By using a component based approach, the standard unit-level object model will not readily become obsolete as technological advances occur.

Reuse can be achieved by developing libraries of standard components. The object-oriented feature polymorphism permits substitution of compatible components in a unit. This could enhance future force structuring as units are assembled from different components and their effectiveness analyzed in the simulation. While this type of evaluation is exceptionally beneficial, associations between the components would have to be carefully modeled to capture symbiotic relationships between the components of the units. There is still much needed research to fully realize the benefits promised by standardizing the unit-level object model.

B. AREAS FOR FURTHER RESEARCH

This thesis is the initial research into the development of the standard unit-level object model. It will be reviewed by the Object Management Standards Coordination Committee in October 1997. This object model should be updated to reflect recommended changes from the committee and additions resulting from the completion of work from the standard data and algorithm committees.

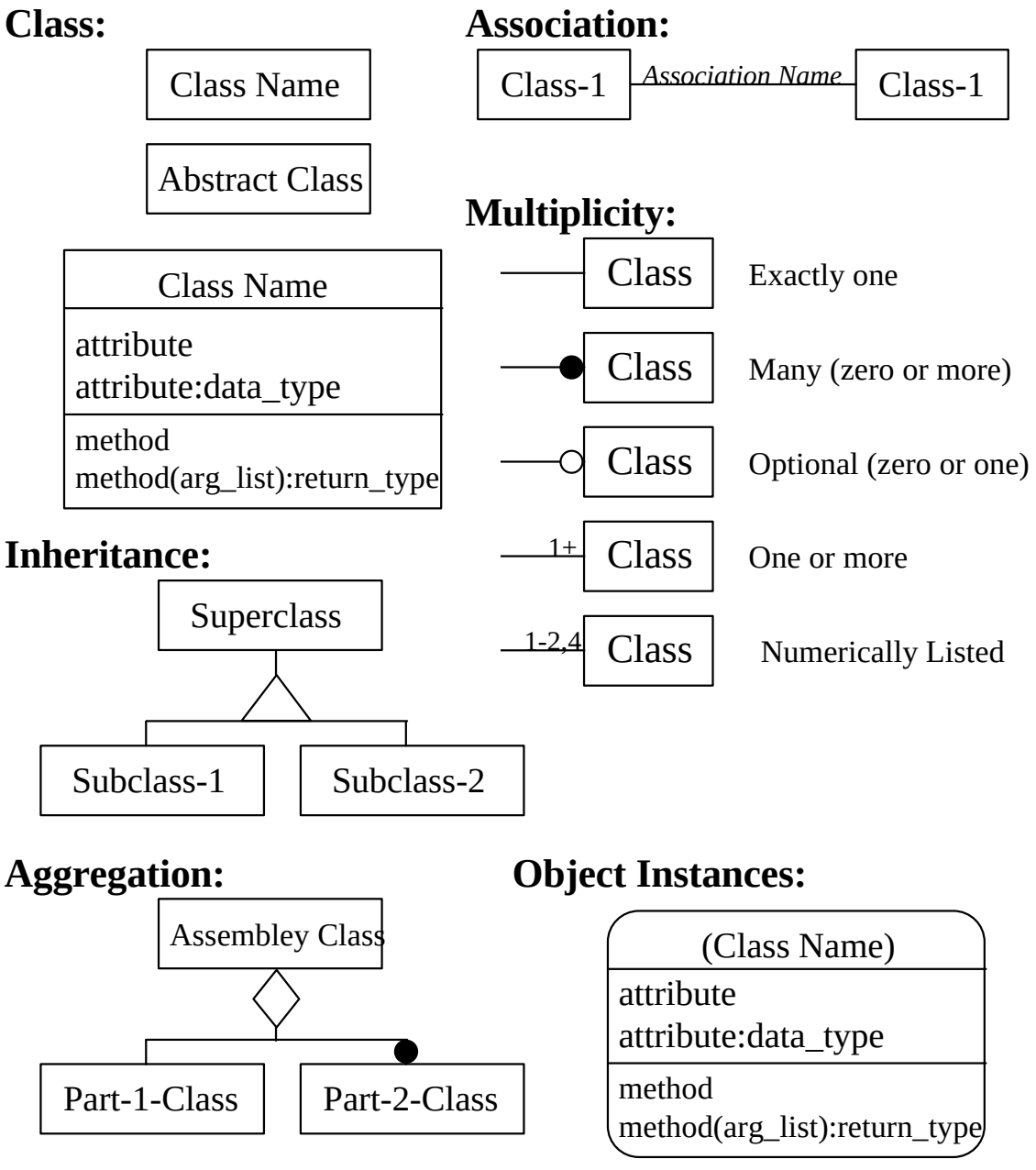
Further research is needed to validate the proposed standard unit-level object model against simulations of all functionalities from all services. The standard unit-level object model should be capable of being implemented as easily as a Naval Inventory

accomplished, in part, by taking a legacy simulation and map it into an object-oriented model based on the standard unit-level and platform-level object models. The standard drawing standardized data from the CMMS and FDB.

Another logical step in the development of standard objects would be to evaluate of a library of standard unit components. Ultimately, the standard unit-level object model will achieve maximum interoperability and reuse by drawing from standard components, updated with minimum effort to reflect future doctrinal, organizational, or technological innovations. Software design is an iterative process and the adoption of standard

APPENDIX

Object Model Notation



Appendix 1

LIST OF REFERENCES

AMSAA, Special Publication No. 77, *Compendium of High Resolution Attrition Algorithms*, October 1996.

Army Modeling and Simulation Office, Standards Category: Object Management, http://www.amso.army.mil/amso2/sp-div/process/obj-mgt.htm.

Blakely, B., McDonald, K., *Functional Description of the Battle Space White Paper*, United States Army Simulation, Training, and Instrumentation Command, 1 April 1996.

Dudgeon, Douglas E., *Development of a Standard Army Object Model*, Naval Postgraduate School, September 1997.

Jackson, Leroy A., Buss, Arnold H., *Standard Army Objects: Interim Report*, 2 September 1997.

JWARS Office, The Joint Warfare System Object Model, 24 September 1996, (http://www.dtic.mil/jwars/library.html)

LORAL Advanced Distributed Simulation Technology Program Office, *ModSAF VOL. 1 (KIT A)*, DIS Service Center, September 1995.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorenson, W., *Object-oriented Modeling and Design*, Prentice Hall, 1991.

Science Applications International Corporation, *Integrated Theater Engagement Model Technical Manual*, February 1995.

Souder, Rich, Walker, Paul, Castner, Ken, McCauley, Bob, *Human Decision Making - Object Oriented Analysis in WARSIM 2000*, Spring Interoperability Workshop, March 1997.

TRADOC Analysis Center, *Eagle Object Model (Advance Copy)*, August 1997.

TRADOC WARSIM Directorate, *WARSIM 2000: Mission Statement and Vision Concept*, http://www-leav.army.mil/nsc/nsc/dis/msn/msn.htm.

Under Secretary of Defense Memorandum, For: Secretary of the Army, Subject: *DoD High Level Architecture (HLA) for Simulations*, 10 September 1996.

INITIAL DISTRIBUTION LIST

No. of copies

1. Defense Technical Information Center.....2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library.....2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Professor Arnold H. Buss, Code OR/BU.....2
   Operations Research Dept.
   Naval Postgraduate School
   Monterey, California 93943-5101

4. Director.....2
   U. S. Army TRADOC Analysis Center-Monterey
   Monterey, California 93943-5101

5. Major Leroy Jackson.....2
   U.S. Army TRADOC Analysis Center-Monterey
   Monterey, California 93943-5101

6. Lieutenant Commander Arthur L. Cotton, III….....2
   204 West Hamilton Street
   Oberlin, Ohio 44074

7. Captain Douglas E. Dudgeon……..……..1
   1817 Leisure World
   Mesa, Arizona 85206

8. Army Modeling and Simulation
   Office……..4
   Attn: Standards and Policy Division, MAJ Johnson
   Crystal Gateway 5, Suite 503E
   1111 Jefferson Davis Highway
   Arlington, Virginia 22202